Spring 2017

Workload Prediction for Efficient Performance Isolation and System Reliability

Ji Xue
College of William and Mary - Arts & Sciences, jxue02@email.wm.edu

Workload Prediction for Efficient Performance Isolation and System Reliability

Ji Xue

Williamsburg, VA

Bachelor of Engineering, Beihang University, 2012

A Dissertation presented to the Graduate Faculty
of The College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

College of William & Mary
May 2017

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Ji Xue

Approved by the Committee, May 2017

Committee Chair
Professor Evgenia Smirni, Computer Science
The College of William and Mary

Professor Weizhen Mao, Computer Science
The College of William and Mary

Associate Professor Gang Zhou, Computer Science
The College of William and Mary

Assistant Professor Xu Liu, Computer Science
The College of William and Mary

Dr. Arif Merchant, StorageAnalytics Team
Google

ABSTRACT

In large-scaled and distributed systems, like multi-tier storage systems and cloud data centers, resource sharing among workloads brings multiple benefits while introducing many performance challenges. The key to effective workload multiplexing is accurate workload prediction. This thesis focuses on how to capture the salient characteristics of the real-world workloads to develop workload prediction methods and to drive scheduling and resource allocation policies, in order to achieve efficient and in-time resource isolation among applications.

For a multi-tier storage system, high-priority user work is often multiplexed with low-priority background work. This brings the challenge of how to strike a balance between maintaining the user performance and maximizing the amount of finished background work. In this thesis, we propose two resource isolation policies based on different workload prediction methods: one is a Markovian model-based and the other is a neural networks-based. These policies aim at, via workload prediction, discovering the opportune time to schedule background work with minimum impact on user performance. Trace-driven simulations verify the efficiency of the two proposed resource isolation policies. The Markovian model-based policy successfully schedules the background work at the appropriate periods with small impact on the user performance. The neural networks-based policy adaptively schedules user and background work, resulting in meeting both performance requirements consistently.

This thesis also proposes an accurate while efficient neural networks-based prediction method for data center usage series, called PRACTISE. Different from the traditional neural networks for time series prediction, PRACTISE selects the most informative features from the past observations of the time series itself. Testing on a large set of usage series in production data centers illustrates the accuracy (e.g., prediction error) and efficiency (e.g., time cost) of PRACTISE.

The superiority of the usage prediction also allows a proactive resource management in the highly virtualized cloud data centers. In this thesis, we analyze on the performance tickets in the cloud data centers, and propose an active sizing algorithm, named ATM, that predicts the usage workloads and re-allocates capacity to workloads to avoid VM performance tickets. Moreover, driven by cheap prediction of usage tails, we also present TailGuard in this thesis, which dynamically clones VMs among co-located boxes, in order to efficiently reduce the performance violations of physical boxes in cloud data centers.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to thank everyone who has helped with this thesis, especially:

My advisor, Professor Evgenia Smirni, who encourages me all the time, continuously teaches me new knowledge, and brings lots of chances to me to grow up in the research world. Without her patient guidance and persistent help, this thesis would not be possible.

My committee members, Professor Weizhen Mao, Professor Gang Zhou, Professor Xu Liu, for their great support and insightful feedback and comments.

My intern and project advisors, Dr. Lydia Y. Chen and Dr. Robert Birke in IBM Zurich Research Lab, Dr. Alexander Shraer from Apple, Dr. Arif Merchant in Google, Dr. Alma Riska in NetApp, and Dr. Jason Hong in Microsoft. Without their guidance and help, I would not go through so many interesting and exciting projects.

All the faculty and staffs at the Computer Science Department of the College of William and Mary. Special thanks to Vanessa Godwin, Jacqulyn Johnson, and Dale Hayes for their considerate and efficient assistance.

Jiawei Wen, Bin Nie, Feng Yan and many other dear friends for their support and help.

Finally, my dear families, especially my parents and loved grandma. They always give me so much encouragement and support. Best wishes to my grandma!

LIST OF TABLES

LIST OF FIGURES

Workload Prediction for Efficient Performance Isolation and System Reliability

Chapter 1

Introduction

In the era of 'big data', from large-scale enterprise and cloud storage systems to data centers, today's systems are highly distributed and shared by many applications. Such resource sharing brings a host of benefits, including efficient resource utilization, low operating cost, improved data availability, and enhanced reliability to name a few. However, resource sharing also faces many challenges. For example, when multiple workloads compete for limited resources, there is potential contention and interference which may not only cause delays in the performance of individual workloads, but also result in inefficiency of resource utilization. The challenges behind resource sharing could be summarized as providing in-time and efficient resource isolation among different workloads, such that for different applications, the corresponding Service-Level-Objectives (SLOs) are met, while for the whole system, resources are well utilized. As a result, for the multiplexing systems, *when* and *how* to isolate resources among different applications matters.

In various systems, including storage systems and networks, a lot of studies have been done to answer the above challenges of resource sharing, such as fairness-driven resource allocation algorithms [102, 18], and balancing trade-off of fairness and performance for different applications, e.g., latency [54] and throughput [114]. In Figure 1.1, an illustration

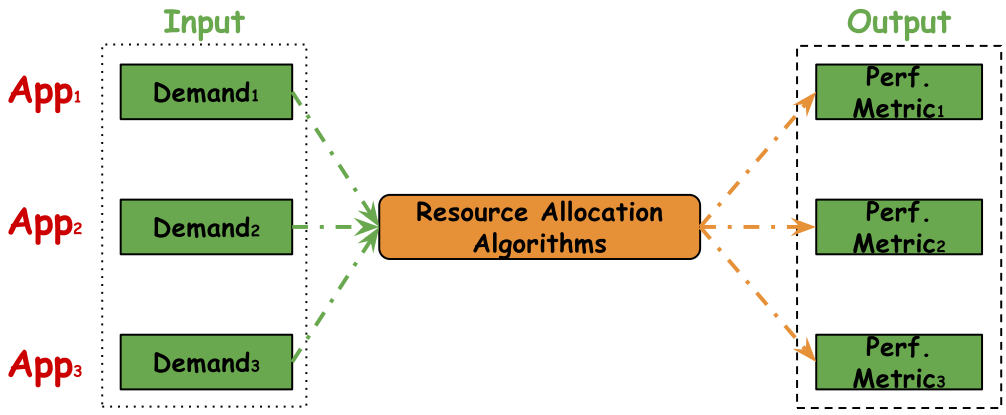
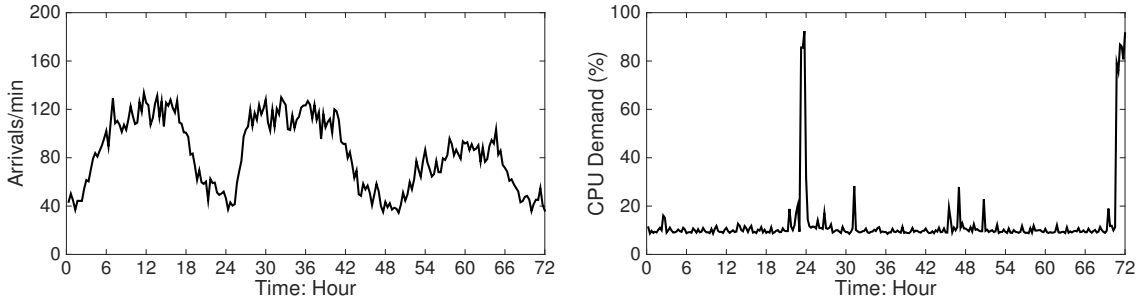


Figure 1.1: An illustration of steps for decisions on resource allocation in the multiplexing systems.

of each step to decide on resource allocations in a system is presented. As shown here, there are two key variable components in the resource allocation decision process: *demands of applications*, and *resource allocation algorithms*. Most of existing work is focused on the latter one, while often assuming perfect knowledge of the future demand for each application or just using the last value prediction. In reality, the demands of applications show fluctuations across time, making the workload prediction non-trivial. In this thesis, we focus on identifying the characteristics of workloads in different systems, proposing prediction methods for them, and incorporating workload prediction into analytical models. Such prediction methods for workloads are critical for performance analytical models and state-of-the-art resource allocation algorithms in these resource-sharing systems.

Workload prediction can be roughly classified as: statistics based and time series based. Statistics based prediction means applying the statistical characteristics of the previous workload, e.g., the probability density function of a metric, to estimate the upcoming workload. Time series based prediction learns the lagged function among the time series itself to directly forecast the upcoming traffic. In today's systems, the user workloads often *fluctuate overtime* (see Figure 1.2). This observation suggests that traditional statistical



(a) Storage System (b) Cloud Data Center

Figure 1.2: Overview of two typical user workloads over three days in two different multiplexing systems. (a) is an example of average arrival intensity of user work over three days in a single data node of a large distributed storage cluster from EMC corporation, here x-axis represents the time in the granularity of hour, and y-axis is the user arrival intensity per minute. (b) stands for a typical VM, with its CPU demands over three days shown, in the IBM production data centers, and x-axis represents the time in the granularity of hour, and y-axis is the CPU demand.

analysis based method fails to capture/predict the *fluctuations* at each *time stamp*. Thus, it is critical to provide time series based workload prediction to forecast the fluctuations of workload overtime, for *in-time* and *efficient* resource isolation. Time series based workload prediction is often deployed into two categories: *coarse-granularity* and *fine-granularity*. *Coarse-granularity* prediction methods, as the name suggests, divide the intensity of workloads into N *states* (e.g., $N = 2$), and then forecast the *state* of the upcoming workloads as well as the corresponding *duration* residing on this state. *Coarse-granularity* workload prediction works for coarse-grained performance requirements of multiplexing workloads, e.g., the mean latency for few hours, but fails in maintaining strict SLOs *consistently*. Thus, there is a clear need for *fine-granularity* workload prediction, which is capable to forecast the intensity of the upcoming workload in a smaller time window W (e.g., $W = 10$ min), so as to timely adjust the input to the resource allocation decision process to consistently meet performance requirements.

One specific target area to apply the workload prediction methods is multi-tier enterprise storage systems which have risen in popularity over the last decades. Multi-tier

storage systems have limited fast-service-rate resources (e.g., DRAM), which are often multiplexed by several application workloads, such as high-priority user workloads, internal workloads for system reliability, as well as data analytics work. The latter two (seen as background work), although labeled as low priority, still affect the performance of user work, especially if they are non-preemptive. Thus, it is critical and difficult to ensure the performance of different workloads and system availability. To address this problem, practitioners use on-line feedback [35] to *reactively* decide on resource isolation. However, if user workload flows in the enterprise storage systems experience drastic changes, such as burstiness, *reactive* resource isolation is too late to maintain user performance. Consequently, it is desirable to *pro-actively* allocate resource among different workloads, by deriving information on the upcoming workloads via time series-based workload prediction.

Another area of interest to leverage the workload prediction methods is cloud data centers, where virtual machines (VMs) are highly consolidated on physical boxes. With physical resources being aggressively multiplexed across multiple VMs, the likelihood of performance violations due to physical or virtual machines crossing predefined thresholds dramatically increases [16]. As a result, developing efficient policies for maintenance on the efficiency of resource multiplexing ratios and avoiding performance violations becomes an important issue in cloud data centers, which can be achieved with the help of time series based workload prediction.

In general, resource sharing in storage systems or cloud data centers, significantly affects system performance and reliability, thus it is necessary to isolate resources in a *pro-active* manner. Additionally, workload fluctuations in such systems create challenges for future workload estimations. By taking advantage of time series-based workload prediction, one can effectively predict the fluctuations for the future workloads and then incorporate them

with the resource isolation decision process, such that limited resources are managed in a pro-active way. This thesis focuses on how to predict future workloads accurately, and make use of time series based workload prediction for achieving in-time and efficient resource isolation in enterprise storage systems and cloud data centers.

1.1 Contributions

The contributions of this thesis are summarized as follows:

- effective *hybrid analytical models* are developed, that predict user traffic and co-schedule user workloads with background work in multi-tier enterprise storage systems [121, 122, 123], see Section 1.1.1;
- a light-weight *time series prediction method* is designed to forecast usage series in cloud data centers [120], see Section 1.1.2;
- efficient *resource isolation polices* are proposed, which efficiently predict the usage series in data centers and proactively reduce performance violations of VMs [118] and physical boxes [119] in cloud data centers, see Section 1.1.3

1.1.1 Hybrid analytical models for performance isolation in storage systems

In multi-tier enterprise storage systems, co-scheduling high-priority user workloads with background works aims at improving the efficiency of resource usage as well as the data reliability, while faces the challenge of performance interference among different workloads. In this thesis, we build hybrid analytical models that make use of time series based workload prediction methods and incorporate with queueing models to predict/isolate the performance of different workloads.

- *Coarse-grained performance isolation* [121]: we propose a Markovian Arrival Process (MAP) [84] model that learns user traffic pattern in the storage systems, and predicts when the user traffic intensity comes into a *high* or *low* state, then *pro-actively* stops/starts bulks of internal system work. We show that this scheduling can result in in-time isolation between user workloads and internal system work, compared to existing on-line feedback methods.

- *Fine-grained performance isolation* [122, 123]: we design a novel hybrid approach *NeuQ*, that consists of machine learning (ML) and queueing models. *NeuQ* applies neural networks to predict the user workload intensities with fine-granularity and then appropriately adjusts the input to a queueing model, in order to consistently meet user SLOs while completing data analytics work as much as possible. Trace-driven simulations confirm the efficiency of *NeuQ* in providing robust user and data analytics work performance, with strict SLOs.

1.1.2 Prediction of data center time series

In cloud data centers, *pro-active* resource isolation among different VMs can effectively provide guaranteed performance to multiple users timely. Key to achieving this goal is to accurately predict physical and virtual machines' usage patterns, especially peak loads in these usage series. In this thesis, we propose a light-weight time series prediction method that can predict the usage series in cloud data centers.

- *Usage series prediction* [120]: We propose a neural networks-based time series prediction method, called *PRACTISE*, to forecast usage series of physical and virtual machines in cloud data centers. We first find the dominant patterns in data center usage series, using the autocorrelation function. Then, we select the most informative features for usage series prediction as input to neural networks. We show that

the proposed neural networks-based time series prediction method achieves better accuracy and efficiency, comparing with existing ARIMA/ARMA and vanilla neural networks.

1.1.3 Active resource isolation in data centers

In today's data centers, physical servers are highly consolidated by VMs. This improves the efficiency of physical resource usage, but also brings the challenge of resource contentions. Resource contentions result in performance violations and reduce system availabilities to users. In this thesis, we develop an active sizing algorithm that is based on an efficient usage workload prediction method and derives virtual resource allocation among co-located VMs, such that performance violations for VMs are reduced or avoided. As well, in order to avoid the performance violations in the physical boxes, we propose a VM cloning policy that captures the relationship between usage mean and tails and proactively clones VMs among co-located boxes.

- *VM performance violation avoidance* [118]: We propose an active sizing algorithm for co-located VMs in cloud data centers, called *ATM*, deriving from a usage series prediction algorithm, that captures the temporal and spatial dependency within/across resources and co-located VMs. The proposed sizing method is shown to be able to reduce the performance violations effectively in production data centers.

- *Box performance violation avoidance* [119]: An efficient VM cloning policy is proposed for co-located physical boxes within the same tenant in cloud data centers, called *TailGuard*, that discovers the hidden relationship between usage mean and tails of boxes, and then proactively drives dynamical VM cloning among co-located boxes. The proposed VM cloning policy is shown to efficiently reduce even avoid the performance violations in physical boxes.

1.2 Organization

This proposal is organized as follows. In Chapter 2, we present an overview of basic concepts and terminology that are used in this thesis. In Chapter 3 and 4, we introduce two different workload prediction methods for pro-active performance isolation in storage systems [121, 122, 123]. Followed by Chapter 5, a neural networks-based time series prediction method is illustrated for resource usages in cloud data centers [120]. Chapter 6 and 7 demonstrate an active resource isolation policy for co-located VMs [118] and boxes [119] in cloud data centers. Lastly, Chapter 8 describes the research plan for the future work in details.

Chapter 2

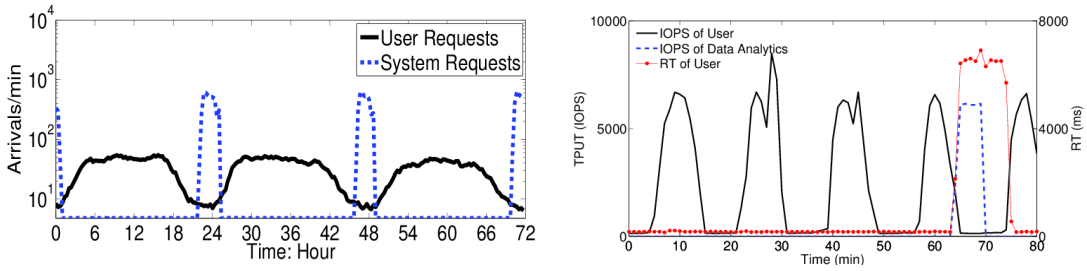
Background

In this chapter, we introduce basic concepts and models that are used in the entire thesis to forecast workloads in systems. We also illustrate the performance impact of resource sharing, which motivates this thesis.

2.1 Introduction to Time Series Prediction

Time series prediction is the use of models that forecast future values of time series variables by extrapolating trends and patterns in the past values of the series or by extrapolating the effect of other variables on the series. In general, based on different granularity requirements, time series prediction is often divided into two categories: *coarse-granularity* and *fine-granularity* prediction. In the following, we present two examples to illustrate the intuitions behind the different-granularity predictions.

Today's enterprise storage systems need to generate and finish internal system work to enhance the data availability and system reliability. Since internal system work in such systems is not instantaneously preemptive, it is desirable to find the most opportune period (namely the period with low user arrival rates) to schedule system work such that



(a) *Coarse-granularity* works (b) *Coarse-granularity* fails

Figure 2.1: Using *coarse-granularity* time series based workload prediction for two different scenarios.

the impact on the user performance is minimum. Figure 2.1(a) gives an example of average arrival intensity of user work over three days (shown as the solid lines in the figure) in a single data node of a large distributed storage cluster. Here the x-axis represents time and the y-axis stands for the arrival intensity per minute. In this figure, we mark the most opportune period to schedule system work with the dotted lines. This example clearly clarifies the need to predict the workload intensity in a *coarse-granularity* manner: using *high* or *low* states.

Coarse-granularity resource isolation cannot guarantee the user performance consistently, an counter example, that applies *coarse-granularity* time series prediction for co-scheduling user workloads with data analytics work resulting in severe user performance violation, is shown in Figure 2.1(b). For this example, we use `fio` [4] as the IO workload generator and generate two types of IO workloads, user and data analytics. The intensity of user IO requests emulates very closely the load pattern of user requests presented in Figure 2.1(a). Note that during the first 60 minutes without data analytics work, user response time remains in the range of 150 ms. While adding data analytics works in the *low* state of user workload intensity (which could be predicted via *coarse-granularity* prediction models) at around the 65th minute, we observe that the user response time immediately increases by two orders of magnitude. This phenomenon suggests that with the require-

ment of strict SLOs for different applications, coarse-granularity resource isolation cannot satisfy the performance requirement. In this scenario, the fine-granularity time series based workload prediction is preferred as input for the resource isolation decision process.

Next, we describe the techniques to achieve the above two different-granularity time series predictions in detail.

2.1.1 Coarse-granularity Time Series Prediction

Coarse-granularity time series prediction, often divides the observed time series into several *states*, and then forecasts *which state* the upcoming series belongs to and *how long* the series stays in the predicted state. In this thesis, Markovian Arrival Processes (MAPs) are used to express/predict the user traffic in coarse granularity. MAPs, introduced by Neuts [84], can easily model and predict any kind of time series with coarse granularity, such as arrivals, service, or even response times. Previous work in [10, 64, 42, 24] has developed efficient schemes to fit and model different time series workloads in systems, approximating short-range and long-range dependence.

A MAP divides the observed time series into N states (based on either requirements or characteristics of the series [81]), and then provides the exponential intensity/rate of each state, together with the jumping probabilities of *background* and *completion* transitions within/across states [21]. A MAP can be specified by two square matrices $(\mathbf{D}_0, \mathbf{D}_1)$, where $\mathbf{D}_0$ describes the transitions across states without signifying any real event while $\mathbf{D}_1$ captures all the transitions associated with real events in the MAP. For example, the schematic of a 2-state MAP is shown in Figure 2.2, with the following 2×2 matrices of $(\mathbf{D}_0, \mathbf{D}_1)$:

$$\mathbf{D_0} = \begin{bmatrix} -\lambda_{1,1} & \lambda_{1,2} \\ \lambda_{2,1} & -\lambda_{2,2} \end{bmatrix}, \qquad \mathbf{D_1} = \begin{bmatrix} \mu_{1,1} & \mu_{1,2} \\ \mu_{2,1} & \mu_{2,2} \end{bmatrix} \tag{2.1}$$

where $\lambda_{1,1}$ equals to $(\lambda_{1,2} + \mu_{1,1} + \mu_{1,2})$, and $\lambda_{2,2}$ is equal to $(\lambda_{2,1} + \mu_{2,1} + \mu_{2,2})$. In $\mathbf{D_0}$, $\lambda_{i,i}^{-1}$ is

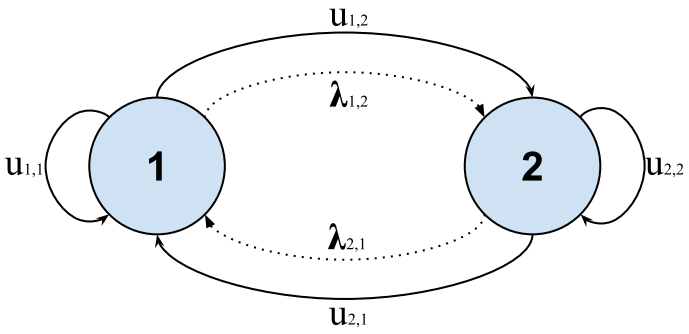


Figure 2.2: State transitions of MAP(2). Solid arrows relate with events transition in the MAP, while dashed arrows associates with the state change only.

the mean time spent in state i before a jump; while $\lambda_{i,j}/\lambda_{i,i}$, with $i \neq j$, is the probability of a *background* transition from the current state i to state j. Similarly, $\mu_{i,j}/\lambda_{i,i}$ represents the probability of a *completion* transition from state i to state j, specifying $\mu_{i,i}/\lambda_{i,i}$ as the probability of returning instantaneously to state i with a completion.

2.1.2 Fine-granularity Time Series Prediction

Different from *coarse-granularity*, *fine-granularity* prediction is able to predict the values of the upcoming time series in each small time window W (e.g., $W = 10$ min). Traditional time series models such as ARIMA/ARMA [23] and Holt-Winters exponential smoothing [51] are limited by the linear basis function, while neural networks are able to model non-linear functions. As a result, in this thesis, we leverage neural networks to predict time series in fine granularity.

An artificial neural network consists of multiple nodes, or *neurons*, which are interconnected to mimic a biological neural network [55], shown in Figure 2.3. These neurons have adaptive *weights*, tuned by a learning algorithm, which enables neural networks to approximate non-linear functions of their inputs. The adaptive weights describe the connection strengths between neurons, activated during training and prediction. Taking advantage

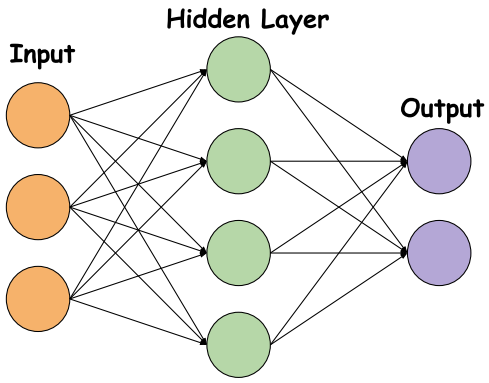


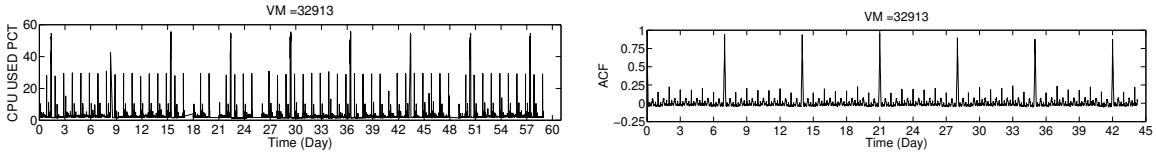
Figure 2.3: An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

of the capability of modeling non-linear functions of the input, we use neural networks to capture the hidden and complex characteristics lying within the time series itself, and then to predict the upcoming series.

Neural networks for time series prediction consist of two steps: *training* and *prediction*. At the first and the most important stage – training neural networks, the training data set composes of the target variables and their corresponding inputs (also named as 'features'). Specifically, for time series prediction, if X is the target variable for prediction, and X_t is the value of X at time t, then at the training stage, the goal is to train a neural network model f of the form:

$$X_t = f(X_{t-m_1}, X_{t-m_2}, X_{t-m_3}, \ldots, X_{t-m_n}) + \varepsilon_t \tag{2.2}$$

where X_{t-m_i}, with $i \in [1,n]$, represent the observations in the time series with m_i time units ahead of the target value X_t, and ε_t is the error term. To obtain a model with high accuracy, X_{t-m_i}, with $i \in [1,n]$, should be informative for predicting the target value X_t. With the neural networks trained at this stage, we are able to predict the upcoming series in fine granularity at the second stage, with the informative features from the observed time series



(a) VM CPU Usage Series (b) ACF of VM CPU Usage Series

Figure 2.4: Illustrating the autocorrelation of a representative resource usage series of one VM in the cloud data center.

as the input.

With the description in this section, one can make use of the above techniques to predict the future series in either coarse or fine granularity. While for fine-granularity time series prediction using neural networks, as mentioned above, plugging the informative features is the key to achieving accurate time series prediction models. In the following section, we describe one statistical measure that identifies the informative features for the neural networks model.

2.2 Autocorrelation

The *autocorrelation* function (ACF) is a mathematical representation of the degree of similarity in a time series and a lagged version of itself [63]. Given a stationary time series $\{X_t\}$, where $t \in \mathbb{N}$, autocorrelation ρ_k is defined as:

$$\rho_k = E[(X_t - \overline{X})(X_{t+k} - \overline{X})]/\sigma^2 \tag{2.3}$$

where $\overline{X}$ and σ^2 represent the mean and variance of $\{X_t\}$, respectively; the index k is called the *lag*, which denotes the number of observations that separate X_t and X_{t+k}. The autocorrelation metric is in the range of $[-1,1]$. Higher positive values indicate that the two points between the computed lag distance are 'similar', i.e., have stronger correlation. Zero values suggest no periodicity. Negative values show that the two points lag elements

apart are diametrically different. As such, it is ideal to discover repeating patterns by quantifying the relationship between different points of a time series as a function of the time lag, via the autocorrelation function. In Figure 2.4, we present CPU usage series of a certain VM in the cloud data center and its autocorrelation function. Visual inspection of Figure 2.4(a) indicates the weekly pattern of this VM on its CPU consumption, while Figure 2.4(b) quantifies the weekly pattern given that the autocorrelation values peak at the lags equal to multiple weeks.

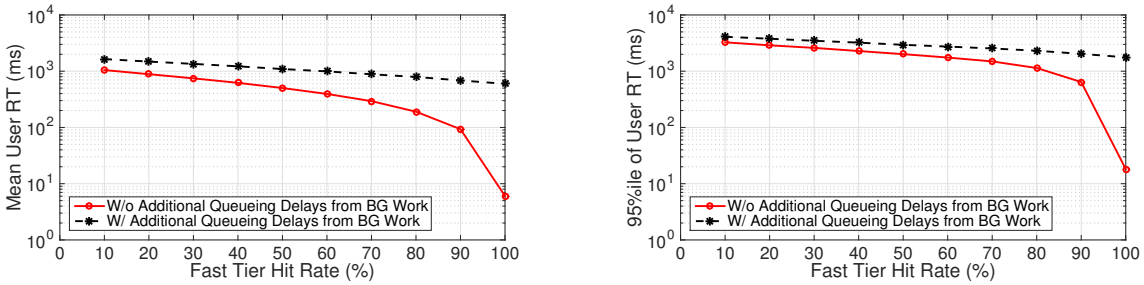
In summary, high absolute autocorrelation values suggest the strongly positive or negative similarity between two lagged points in the time series. In this case, one of these two points could be used as an informative feature for predicting the other. As a result, applying the autocorrelation function allows us to find the informative features for the neural networks based *fine-granularity* time series prediction model.

2.3 Performance Impacts

After introducing the basic techniques for time series prediction, in this section, we will take two simple examples to exemplify the performance impact from resource sharing in the multi-tiered storage systems and cloud data centers.

2.3.1 Performance degradation when co-scheduling

To illustrate the impact on the user performance, e.g., response time (RT), when co-scheduling user workloads with background work (e.g., system work) in multi-tier storage systems. Without compromising on the generalization of the concept, we consider a two-tier form as the test case with around 100× service rate difference between the fast and slow tiers. We implement simulations that consider a time frame of a day. Here the mean user arrival rate equals to 50 requests/min and the mean fast tier service rate is 10000



(a) Mean: w/ Additional Queueing Delays (b) Tail (95%ile): w/ Additional Queueing Delays

Figure 2.5: User performance impacts from scheduling background work.

(user requests)/min or 100 (system requests)/min, all following exponential distributions. Via simulations, we find that scheduling background work impacts user performance for the following two reasons: 1) decreasing the fast tier hit rate for user workloads, and 2) aggravating additional queueing delays.

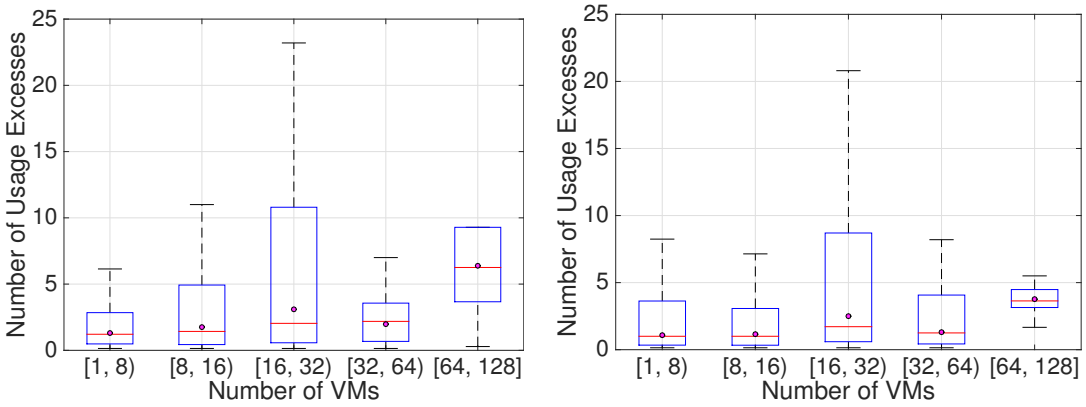
First of all, we briefly discuss on the effect from fast tier hit rate change for user workloads. When scheduling background work, due to the limited capacity of the fast tier, the user working set is evicted from the fast tier. As a result, the fast tier hit rate for the user workloads decreases when scheduling background work. We evaluate the mean and tail user RT with the fast tier hit rate changes only, shown as the solid lines in Figure 2.5. Notice that, here we directly test different fast tier hit rates rather than introduce additional queueing delays from scheduling background work. It is clearly shown that higher fast tier hit rate results in better user performance, for both mean and tail (95%ile) RT, especially comparing the two extreme cases, i.e., fast tier hit rates equal to 10% and 100%. This result confirms that scheduling background work impacts user performance in terms of changing the fast tier hit rate for user workloads. Moreover, we do another set of simulations, which introduce additional queueing delays from scheduling background work. Here we only schedule background work during the idle time. Due to the non-preemptive aspect of background work, scheduling background work during

the idle time still results in additional queueing delays for the user workloads. Mean and tail user RT with different fast tier hit rates, are presented as the dashed lines in Figure 2.5. Comparing the two lines in Figure 2.5(a), for the same fast tier hit rate, we can see an increase in mean user RT when additional queueing delays are calculated. Similar results are derived for the tail user RT from Figure 2.5(b). This illustrates that scheduling background work impacts user performance in terms of aggravating additional queueing delays to user workloads. In summary, co-scheduling in the multi-tier storage systems has a clear impact on user performance.

2.3.2 Resource Contention in data centers

To demonstrate the performance impact from resource sharing on today's data centers, we conduct an post-hoc characterization study on traces of $\sim$ 6K physical boxes in IBM production data centers, containing CPU usage series over one week. Previous work [28] has linked the performance violations with over utilizing resources in cloud data centers. While the number of VMs in the physical boxes (namely the consolidation level) represents the level of resource sharing in the physical boxes, to some extent. As a result, to illustrate the performance impact from resource sharing in cloud data centers, we explore the relationship between the consolidation level and resource usage across all physical boxes.

In Figure 2.6, we present the boxplots that show the 25^{th}, 50^{th}, and 75^{th} percentiles (boxes), the extremes of the distributions (whiskers) and the means (dots) of the number of CPU usage excesses for different ranges of VM consolidation levels on the x-axis. Here a usage excess represents the usage being above the pre-defined usage thresholds (e.g., 60% or 70%) [16]. It is clear to see that higher VM consolidation levels result in more CPU usage excesses, comparing the mean (circles in the figure) for each consolidation level. The above result suggests that more VMs share the physical resource, higher probability to have



(a) Usage Threshold = 60% (b) Usage Threshold = 70%

Figure 2.6: Average number of box CPU usage excesses per day, across 6K different physical servers with different consolidation levels.

usage violations. In conclusion, resource sharing in cloud data centers introduces incurring performance violations caused by resource sharing.

2.4 Chapter Summary

In this chapter, we give an overview of basic concepts and terminology. On the one hand, we focus on the time series prediction, the common theme in this thesis, as well as the related techniques, including Markovian Arrival Process, neural networks and the autocorrelation function. In particular, we give an overview of the Markovian Arrival Process to predict the user workloads in coarse granularity (see Chapter 3) and neural networks to forecast the user traffic in a much finer way (see Chapter 4). The autocorrelation function is used to find the periodicity pattern in the time series, in order to determine the most informative features for the neural networks based time series prediction (see Chapter 5). One the other hand, we target at the performance impact of resource sharing, the main application scenario in this thesis, for both storage systems (see Chapter 3-4) and cloud data centers (see Chapter 6-7). In summary, this thesis applies different time series based workload

prediction methods that can be incorporated with resource isolation and scheduling polices, such that the guaranteed performance and system reliability are achieved.

Chapter 3

Coarse-granularity Performance Isolation

In this chapter, we focus on the performance isolation in the multi-tier storage systems based on coarse-granularity workload prediction. As data storage technologies such as flash make their way into either enterprise storage [89, 62] or cloud storage [1], it is essential to integrate them with existing and (usually slower and cheaper, even vintage) ones, e.g., hard disk storage, in order to strike a good balance among overall performance, availability, and cost [86]. Following tradition in costs across the data path of a computer system, the fastest data tier, e.g., DRAM, is the most expensive, while the slowest tier, e.g., hard disk storage, is the least expensive per unit of data stored. In high-performing enterprise storage systems, it is common to find large DRAM caches, reaching as much as hundreds of GByte capacity, SSD caches reaching tens of TBs capacity, and a variety of high-end HDDs (15KRPM SAS HDDs) and low-end HDDs (7200RPM Nearline-SAS HDDs), all together exceeding the PByte-range of storage capacity. Given such a structure in the IO hierarchy, the expectation is that the bulk of user workload is served by the fast tiers, while slow tiers are used to persistently store the majority of data as well as improve on storage capacity

and data reliability. The challenge lies with determining *what* portion of the voluminous data set to bring up to the smaller fast tiers and *when* to do that such that the benefits with regard to user performance and overall system operation are the highest.

Storage systems, both enterprise ones and those supporting web services, often receive the bulk of user traffic during business hours on weekdays. In addition, the storage system generates itself a considerable amount of internal traffic as a result of complex features that aim to enhance performance, reliability, availability, and integrity. Such system internal work includes, but is not limited to, making additional copies of the data off-site for added disaster recovery capabilities, snapshooting, deduplication, and policy compliance in a multi-tenant system. Recently, other sources of work have emerged in scaled-out storage systems such as virtual data analytics clusters that are brought up on demand to conduct analysis on large data sets without moving the data to a separate compute cluster, requiring effective interleaving of user and system workloads. As discussed in Chapter 2, during the day, user requests peak in intensity at around noon. During night, at regular intervals, the system generates its own work, which clearly is bulky and would have impacted user performance significantly if not scheduled during off-peak hours.

Because the system work is the result of several features, it greatly surpasses in intensity all other user traffic. More importantly, its working set is (usually) much larger than the user working set. As a result, in a well balanced multi-tiered system, system work could negatively impact data placement policies which ensure that the most active working set is in the highest performing tier. To schedule system work at regular intervals (e.g., at around midnight) does not necessarily guarantee good user performance (note that the figure illustrates only the arrival intensity, not work that needs to be done). Here, we contend that it is necessary to pair scheduling of system work with other system metrics measuring user activity and workload, such as utilization due to user-traffic, user traffic

intensity, and fast storage tier hit rates, in order to improve overall system usage while ensuring that system work completes timely.

A critical difference between system internal work and user traffic is that the corresponding working sets are vastly different in both footprint and location within the storage system. Generally, the system working set is much larger than the user working set. As a result, in a multi-tiered storage system with tiers having different capacities and performance characteristics, standard efforts to isolate system and user workload in-time may still result in poor user performance. As the system transitions between system work to user work, high performing tiers could experience high user miss rates. Warming up the faster tier with the user working set is not instantaneous [128] and unless done proactively performance degradation of user traffic can become unsurmountable.

In this chapter, we propose an autonomic technique that over time learns the intensity patterns of user work within a probabilistic model over long time-scales, i.e., days and hours. The prediction of user intensity is paired with the knowledge of tier capacity, performance differences across tiers, and other metrics such as active user data set to derive a schedule for system work, i.e., *when* to start and stop it. Predicting user intensity patterns at large time scales can support

- proactively stopping system work *before* the user intensity increases and warming up the fast tier with the user working set,
- scheduling system work according to predicted user activity patterns such that large amounts of system work is completed with minimal impact on user performance, and
- avoiding instability due to short-lived, low user intensity that may erroneously initiate voluminous system work if short time-scale prediction or if reactive (i.e., feedback-based) scheduling is used.

Our methodology is light and robust. Its benefits are evaluated via trace-driven simulation and actual experiments on a real test-bed. We do comparisons against feedback-based techniques that are usually applied in such settings. Our experiments indicate that the larger the fast tiers and the larger the active user working set, the higher the benefits of having predictive models to schedule bulky system work.

This chapter is organized as follows. In Section 3.1, we discuss on the related work. Section 3.2 presents a workload characterization of user workload in a storage system supporting web data services. In Section 3.3, we present our predictive algorithm. Section 3.4 presents experiments on a real system and trace driven simulations that demonstrate the effectiveness of our technique. We conclude this chapter in Section 3.5.

3.1 Related Work

There is a rich body of work in the literature on storage tiering. Hierarchical storage systems are early examples of storage tiering techniques . HPSS [7] has higher tier (disk) and lower tier (tape) but only allows files to be read or written from the higher tier while the lower tier is treated as an offline device, e.g., data must first migrate to the higher tier before been accessed. VxFS [44] improves the flexibility of the early hierarchical storage systems by allowing user defined placement and migration rules. As the cost of SSDs reduced, SSDs have been introduced in the storage hierarchy. HP'S 3PAR [89] and EMC's FAST [62] are examples of such systems.

Storage tiering is usually critical for meeting service level agreements (SLA) because it can significantly boost the overall system performance. Amazon provides ElastiCache [1] for improving application performance by adding an in-memory caching layer to the infrastructure. FlashTier [96] proposes an interface that is designed for using an SSD as a fast storage tier. Everest [82] offloads bursty I/O workloads by using spare disk bandwidth

to a virtual short-term persistent storage so that the I/O request latency during peaks is improved. There are other storage tiering works focused on improving reliability [8, 87, 25] and cost or energy savings [53, 86].

There are various system storage works that are usually scheduled as a "background" activity for various purposes, including replication [99, 113], security [67], and data analysis [108, 80]. Several workload interleaving techniques [39, 124] have been proposed for scheduling such system or background work, but they do not consider storage tiering.

The work most related to ours is optimizing storage cache warm up. Bonfire [128] accelerates the cache warmup by using more efficient pre-load methods. Windows SuperFetch [5] pre-loads the frequently used system and application information and libraries into memory based on the usage pattern in history to reduce the system boot and application launching time. While Bonfire [128] and SuperFetch [5] focus mostly on identifying the data locations that should be brought into the fast tier for higher efficiency, our work concentrates into identifying *when* to proactively bring a specific and pre-identified data set into the fast tier such that the system can be best utilized by system work with minimal impact on user perceived performance. In this regard, our proposed predictive framework can be viewed as complementary to Bonfire [128] and SuperFetch [5].

3.2 Trace Overview and Motivation

In this section, we present a set of production traces that describe how a storage system is utilized from a large scale web application. The traces contain the user IO intensity (in requested files per second) in a scale-out storage back-end of a mid-size web service provider[1]. The web service has multiple locations that serve the user workload based on geography. As a result, in each location the workload intensity follows well the day/night

[1]Due of confidentiality agreements, the traces or provider details can not be made publicly available.

pattern (working hours vs. non-working hours) as well as weekday/weekend patterns. Here we focus on the traffic received by a single data node. However, because of the load balancing in the storage system, the behavior observed in a single node persists across all other data nodes in the cluster.

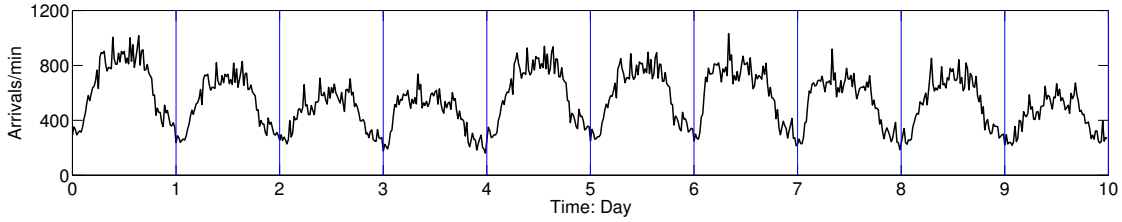


Figure 3.1: User request arrival intensity (number of arrivals per 10 minutes) over 10 days.

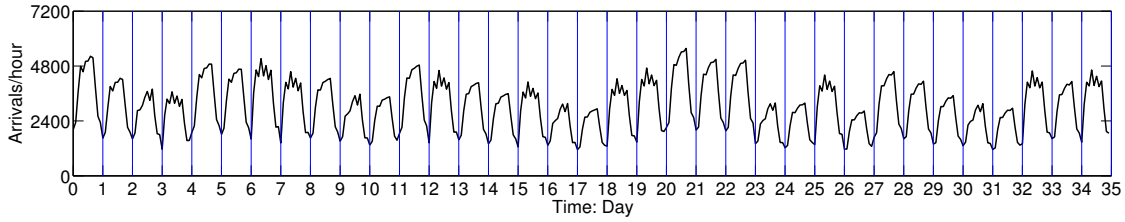


Figure 3.2: User arrival intensity (number of arrivals per hour) over 35 days.

Figures 3.1 and 3.2 show the average arrival intensity of user requests per minute averaged over 10 minutes and 1 hour intervals for 10 days and 35 days periods, respectively. There is a clear daily and weekly pattern in the workload intensity. This is expected since nowadays large scale web services, although available 24/7 worldwide, are deployed in geographically distributed data enters, resulting in clear day-and-night patterns in each of the available locations. Similar patterns are seen also in enterprise storage, which although different in nature from web storage, serves heavy traffic during business hours and much less during night hours. These patterns suggest opportunities for predicting user traffic intensity. The ability to predict these drastic changes can be used to prepare the system proactively for the heavy user workload, for example by moving the active user data set

to the fast tier *before* it starts being accessed. Effective prediction should also ensure that the system schedules long and resource demanding internal work only when it is safely predicted that the system is to enter a long period of low utilization.

We plot the empirical density of the user arrival rate at a granularity of a minute in Figure 3.3 for all 35 days. The graph illustrates a clear bi-modal pattern, which confirms that the arrival intensity changes between two general states that we roughly classify as high/low. In the next section, we show how we incorporate the stochastic characteristics of the user arrivals to derive a model that predicts the duration or each high and low intensity periods.

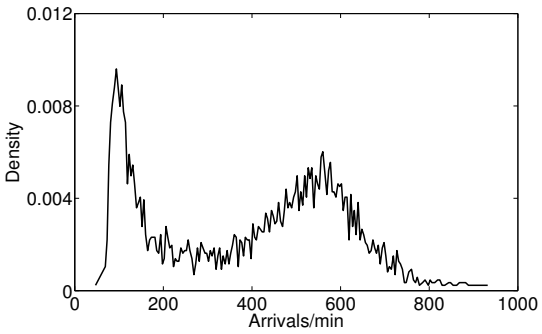


Figure 3.3: Empirical density of the arrival rate of user requests.

Prediction of low/high intensity user periods provides the system with the information it needs to intelligently interleave user workload with voluminous system internal work. For example, it can proactively stop the system work and warm up the fast tier storage just before the high user intensity period so that the majority of user traffic is served by the fast tier rather than the slow tier. As discussed later in the chapter, we do not determine here what data to bring but rather when to bring them in the fast tier.

To illustrate the benefits of predicting arrivals of high and low intensity periods and motivate our work, we evaluate three scenarios on handling system work in a data node:

- *user only*: no system work in interleaved with user traffic,

- *reactive*: system work runs only during low user utilization periods; when user high utilization is detected, the system work is stopped and reactively the fast tier cache is warmed up with active user data,

- *proactive with future knowledge*: system work runs only during low user utilization periods; since we know *a priori* when user high utilization starts, we stop system work early enough to allow for the fast tier cache to be warmed up with active user data right before the surge of user work.

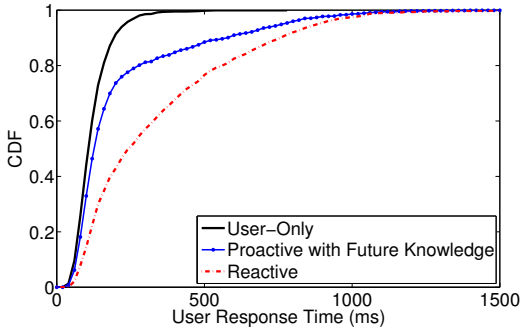


Figure 3.4: CDFs of user response time of day 20 for different algorithms.

Figure 3.4 illustrates the CDF of user response time when a single day worth of trace data (see Figure 3.2) is used to drive a simulation of the above three scenarios. For the two policies that allow system work, the simulation starts and stops it at the same time, with the purpose of evaluating only the benefit of proactive vs. reactive fast tier warm up (which takes the same amount of time in both scenarios). Clearly with a reactive warm up a large portion of user requests experience long response time by being served from the slow tier of the system. Both the body and the tail of the user response time distribution benefits greatly by a proactive fast tier warm up, which can be is possible only if a model enables prediction of arrival of high user utilization periods.

3.3 Model-based Storage Tiering

The data patterns described in Section 3.2 (see Figures 3.1 and 3.2) favor prediction. User intensities go through a clear high/low pattern which if captured accurately can be used to intelligently interleave workloads with widely different demands. Indeed, this is the current state of the practice in most data centers [16]. We aim to develop a model that would allow for better overall system resource utilization by completing aggressively system work and achieve better performance isolation across user and system workloads.

We first present a Markovian-based model that captures the duration of low/high traffic intensities in user arrivals across different time scales (i.e., daily distinguishing between weekday/weekend and hourly distinguishing between day/night activity). We also develop a model that captures the changes in user performance as a function of fast tier hit rate. Finally, we apply these models to predict when such periods of high/low intensities arrive to schedule system work and cache warm up with the goal of optimizing performance.

3.3.1 Traffic Intensity Prediction Model

The preliminary workload analysis in Section 3.2 showed that there are repeatable low/high daily intensity patterns. We refer to the state with high average arrival intensity as the *High* state and the lower one as the *Low* state. The threshold for distinguishing the *High* and *Low* states can be discovered via statistical analysis. Alternatively, the threshold could be user-defined. We need to determine the following: how long does the user traffic resides in each state, i.e., the *duration* of each state and the conditional probability that there is a transition from one state to the next. The analysis of Section 3.2 also showed that weekend *High* state intensity is different from weekday *High* state intensity. We aim to capture these patterns in order to distinguish days with overall less intensity from days of higher intensity.

To capture this effect, we use a hierarchical model that captures different *types* or *classes* of high/low intensities. The difference between these is that the average intensity for the *High* or *Low* states may be different as well as the duration of each of these states. Note that in addition to transitions within the *High*/*Low* states within each class, there are probabilistic transitions from the states that represent one class (e.g., weekdays) to another class (e.g., weekend or holidays). This hierarchical model is shown in Figure 3.5. The model in Figure 3.5 has two classes of high/low intensities (capturing day/night and weekday/weekend patterns). However if more classes are detected then the hierarchy of the model can grow to accommodate them.

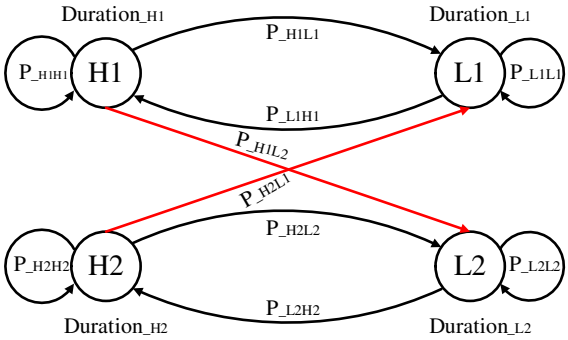


Figure 3.5: High level Markovian model.

We start building the model by first categorizing the observed arrival intensities. We use clustering to determine how many types of low/high intensities exist in the workload using *Silhouette* [94] and *K-means*. *Silhouette* is used to calculate the dissimilarity value $s(i)$ of the average arrival intensity of day i. The dissimilarity value $s(i)$ is defined as:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}},$$

where i is the day index, $a(i)$ is the average dissimilarity of day i to all other days within the same cluster, and $b(i)$ is the lowest average dissimilarity of day i to all days in a different

cluster. Distance measures are the most common for calculating the dissimilarity values $a(i)$ and $b(i)$. The values of $s(i)$ are in $[-1,1]$ and the larger its value the better, e.g., when $s(i)$ approaches to 1, $a(i) \ll b(i)$, which means that the distance between data within each cluster is the smallest. More specifically, the following three steps are performed to determine the number of clusters in the model:

1. Define the upper bound of the number of clusters as $\sqrt{\frac{n}{2}}$, where n is the total number of days in the historical information[2].

2. Calculate the average $s(i)$ for a different number of clusters;

3. Choose the number of clusters with the highest $s(i)$.

After the number of clusters is determined, then we calculate the transition probabilities between them. We also estimate the duration of *High*/*Low* states within each cluster as well as their transition probabilities.

In a live system, the goal is to have an initial model built with the data collected over the first few weeks of operations. Then, the model is updated continuously as new data on user workload is collected, so that any changes in system operation and user access patterns are reflected in the model.

Figure 3.6 illustrates the effectiveness of prediction by comparing it with actual state changes. The dashed lines illustrate the points where the model detects a change in the state (from *High* to *Low* or *Low* to *High*). The dotted line illustrates the actual state changes. The graph shows that the model predicts effectively changes from one user intensity state to the next.

[2] $\sqrt{\frac{n}{2}}$ is used as a rule of thumb in *K-means* to avoid too many clusters and unnecessary overheads [112].

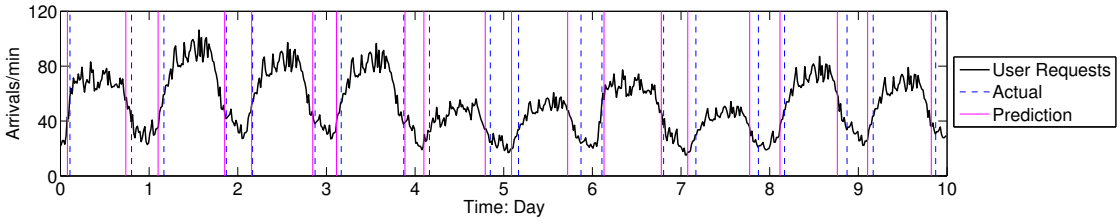


Figure 3.6: Comparison of actual and predicted arrival intensity state changes.

3.3.2 Fast Tier Hit Rate

The fast tier hit rate in a storage system is related to many factors, including its capacity and active user working set. Here, we provide an estimation method for the instant fast tier hit rate with the goal of estimating how it changes as active user data moves from the slow tier up to the fast tier and vice versa.

As we focus mostly on large capacity fast tiers as well as large active data sets, it becomes necessary to warm up the fast tier cache rather than allow it to be warmed up gradually by the user accesses. Figure 3.4 clearly illustrates that warming up the cache can tremendously affect performance.[3]

The average IO service rate for user traffic is a combination of fast storage tier access speed and slow storage tier access speed and can be expressed as follows:

$$\mu(t) = (1 + S(t)) * \mu_{origi} = H * R_{fast} + (1 - H) * R_{slow}, \tag{3.1}$$

where $\mu(t)$ is the average service rate of user traffic at time t. μ_{origi} is the original average service rate of user traffic, e.g., when there is no system work. $S(t)$ is the service slowdown which describes how the average service rate changes from the original one. H is the fast tier hit rate, R_{slow} is the average slow storage tier access speed and R_{fast} is the average fast

[3]The model presented here can be trivially extended to capture the no warm up case, i.e., passively move data when first accessed, by changing the parameter of the average transfer speed between the fast storage tier and slow storage tier to a function that is determined by the intensity of arrivals.

storage tier access speed, implying that the fast tier hit rate can be defined as follows:

$$H = \frac{(1+S(t)) * \mu_{origi} - R_{slow}}{R_{fast} - R_{slow}} \tag{3.2}$$

with $0 \leq H \leq 100\%$.

When system work is served, the average service rate of user traffic unavoidably decreases due to sharing of the fast storage tier with the working set of the system workload. We assume that the Service Slowdown increases linearly over time during the periods of serving system work:

$$S(t) = S(t_{i-START}) + a * t, \tag{3.3}$$

where $S(t_{i-START})$ is the Service Slowdown at the beginning of the time window i serving the additional work. This parameter is necessary because the slowdown effects may propagate through several windows of time. Finally, a is an coefficient that describes how fast the slowdown increases during system work serving periods.

The maximum slowdown occurs when the fast storage tier is filled with the system working set, and unavoidably all user traffic is served from the slow storage tier, therefore

$$R_{slow} = (1+S_{max}) * \mu_{origi}, \tag{3.4}$$

or

$$S_{max} = \frac{R_{slow}}{\mu_{origi}} - 1. \tag{3.5}$$

Note that μ_{origi} may not equal to R_{fast} because the Fast Tier Hit Rate may not equal to 100% even when no system work is served.

When $S(t_{i-START}) = 0$, i.e., when the fast storage tier is filled with the user working set, T_{fast} expresses the period before system work data starts occupying the entire fast storage tier. After the user working set is removed from the fast storage tier, the user service

slowdown reaches its maximum, i.e., no user IO requests can use the fast storage tier to improve performance. According to Eq. 3.3 we have:

$$S_{max} = a * T_{fast}. \tag{3.6}$$

By definition, the capacity C equals to the transfer speed F multiplied by time, therefore:

$$C = F * T_{fast} \tag{3.7}$$

and

$$\mu(t) = (1 + S(t)) * \mu_{origi}. \tag{3.8}$$

From Eq. 3.3 and Eq. 3.5 – Eq. 3.7, when $t > t_{i-END}$, we have :

$$\begin{aligned} S(t) &= S(t_{i-START}) + \frac{S_{max} * F}{C} * t. \\ &= S(t_{i-START}) + \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t, \end{aligned} \tag{3.9}$$

which shows that the user service slowdown is related to the average fast and slow storage tier access speed, the average transfer speed between fast and slow storage tiers, the original average service rate of user traffic, and the fast tier capacity.

When the system stops serving system work, the user service slowdown due to sharing of the fast tier with system workload decreases over time. We assume that this decrease is linear across time:

$$S(t) = S(t_{i-END}) - b * t. \tag{3.10}$$

$S(t_{i-END})$ is the user service slowdown at the end time of system work serving window i and b is an coefficient that describes how quickly the slowdown decreases over non-system work serving periods.

Similarly, when $S(t_{i-END}) = S_{max}$, i.e., when the fast tier is filled with all system work data, it takes T_{fast} time units before the user working set refills the entire fast tier. After the user working set is restored, the user service slowdown reaches its minimum, i.e., to the original service rate without any system work. According to Eq. 3.10, when $S(t_{i-END}) = S_{max}$, $S(t) = 0$:

$$0 = S_{max} - b * T_{fast}. \tag{3.11}$$

By comparing Eq. 3.3 and Eq. 3.10, we have $b = a$. Therefore, for $t_{i-START} \leq t \leq t_{i-END}$, we have:

$$\begin{aligned} S(t) &= S(t_{i-START}) - \frac{S_{max} * F}{C} * t \\ &= S(t_{i-START}) - \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t. \end{aligned} \tag{3.12}$$

Using Eq. 3.9 and Eq. 3.12

$$S(t) = \begin{cases} S(t_{i-START}) - \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t, \\ \qquad \text{for } t_{i-START} \leq t \leq t_{i-END}, \\ \\ S(t_{i-END}) + \frac{F}{C} * (\frac{R_{slow}}{\mu_{origi}} - 1) * t, \\ \qquad \text{for } t > t_{i-END}. \end{cases} \tag{3.13}$$

From Eq. 3.2 and Eq. 3.13 it follows that the hit rate is:

$$H = \begin{cases} \frac{\mu_{origi}(S(t_{i-START})+1) - R_{slow} - \frac{F}{C} * (R_{slow} - \mu_{origi}) * t}{R_{fast} - R_{slow}}, \\ \qquad \text{for } t_{i-START} \leq t \leq t_{i-END}, \\ \\ \frac{\mu_{origi}(S(t_{i-START})+1) - R_{slow} + \frac{F}{C} * (R_{slow} - \mu_{origi}) * t}{R_{fast} - R_{slow}}, \\ \qquad \text{for } t > t_{i-END}. \end{cases} \tag{3.14}$$

3.3.3 Storage Tiering

Figure 3.7 presents an algorithm for scheduling system work in a multi-tier storage system. In the *characterization state*, the algorithm collects arrival intensity information to compute the parameters and build the Markovian model. Based on the *Low* and *High* states duration and the fast tier warm up time, the algorithm schedules system work. For example, during the *Low* state, the system work is served concurrently with the low user traffic because the overall performance impact is small. The thresholds of the fast tier hit rate can be considered a control knob. For example, the thresholds of *Low* state can be set much smaller than the threshold of those of the *High* state so that more system work can be finished.

The algorithm proactively warms up the fast tier by stopping system work ahead of the predicted arrival of the *High* state. The warm up time depends on the fast tier capacity and can be computed via Eq. 3.7. Such proactive action is critical because the fast tier can not be warmed up instantly. For large fast tiers, the warm up may take a long time, hours or in some cases even days [128]. Without proactive warm up, the user requests that arrive in the initial period of the *High* state are to be impacted significantly.

1. if system in *characterization state* do
  a. collect arrival intensity information and use
    Silhouette and *K-means* to do clustering.
  b. Compute duration of *High/Low* states per cluster
    and transition probabilities within and across clusters
  c. Build the Markovian model with the computed
    parameters and continue updating the model
    while the system is in operation
2. if the system is in *serving system work* state do
  a. Predict how long the system will be in *Low* state
    and compute *warmup time* for fast storage tier T_{fast}
  b. if residual time in *Low* state $> T_{fast}$
    i. compute the Fast Tier Hit Rate and
      average utilization $UTIL_{past}$ of past t minutes
    ii. if no outstanding user request
        and $H >= H^{low}_{threshold-lower}$
        and $UTIL_{past} <= UTIL_{threshold}$
        schedule system work
    iii else if $H <= H^{low}_{threshold-lower}$, stop serving
        system work until $H = H^{low}_{threshold-upper}$
      go to Step **2.b.iv**
    iv. else process user request or stay idle
    v. go to Step **2.b**
  c. else if the residual time in *Low* state $< T_{fast}$
    i. stop serving system work and warm up the fast tier
    ii. go to Step **2.b**
  d. else if system in high arrival intensity state
    i. if no outstanding user request
        and $H >= H^{high}_{threshold-lower}$
        and $UTIL_{past} <= UTIL_{threshold}$
        schedule system work
    ii. else if $H <= H^{high}_{threshold-lower}$, stop serving
        system work until $H = H^{high}_{threshold-upper}$
      go to Step **2.d.iii**
    iii. else process user request or stay idle
    iv. go to Step **2.b**
    go to Step **1**

Figure 3.7: Prediction-based deployment of systems work.

3.4 Experimental Evaluation

In this section, we evaluate the proposed scheduling framework via an extensive set of experiments in a real system and through trace-driven simulations. We first describe the testbed and the workload we use in Section 3.4.1 and then show the respective experimental results that validate our method in Section 3.4.1.1. Then we use our traces from Section 3.2 to drive a set of simulation experiments for more sensitivity analysis of our predictive model. Throughout this section we compare our framework with other common practices such as feedback-based techniques.

3.4.1 Experimental Testbed and Workloads

Our testbed consists of a server with a disk enclosure attached to it, which provides data services to a host. Its memory is 12GB and the disk enclosure has 12 SATA 7200RPM HDDs of 3TB each. In our experiments the system memory emulates the fast tier and the disk enclosure the slow tier used for the bulk of the data. The benefits of effective workload prediction are high for system with large gaps in the performance characteristics across tiers. For the shake of presentation clarity, we evaluate here the predictive framework on a system with two-tiers only that provide services that differ by one order of magnitude. We stress that our approach can be directly applied in a system with *any* number of storage tiers.

The workload is generated and measured at the host machine. We use `fio` [4] as the IO workload generator for the flexibility it provides to generate a wide range of IO workload intensities and general patterns. We generate two types of IO workloads, user and system, which differ on the active working set size rather than their access pattern. The working set size for the user workload is 1GB[4], i.e., such that it always fits into the memory of the

[4]Experiments with 4GB and 8GB user working sets yielded similar results and are omitted here due to

server that emulates the fast tier. The system work has an active working set of 24GB, i.e., it does not fit fully into the fast tier and the large slow tier is accessed to retrieve the data. The access pattern for both user and system workload is 100% small random reads to emulate common enterprise workloads that would benefit from prefetching (warm-up) only if the working set can fully (or almost) fit in the high performing tier (i.e., the SSD).

Our framework determines only *when* to warm up the cache with a pre-determined user data set. Determining what data should be brought into the cache is outside the scope. The user active working set can be determined by evaluating statistically access patterns such as the number of accesses per storage location. Here we also assume that the system is provisioned in such a way that the fast tier can fit the entire (or the majority of the) user active working set. The fast tier is warmed up via a sequential read of the user working set.

We have measured the following system work scheduling policies:

- *user-only* - used only as a baseline to evaluate the impact of the additional system work,
- *feedback-based* - a reactive policy that monitors the current load intensity in the system and determines if it is in a high or low intensity period,
- *prediction-based* - a proactive policy (see Figure 3.7) that uses the proposed Markovian model to predict user traffic intensity by having learned from past data the duration of periods of high and low intensity.

Rules that determine the change of state (from *High* to *Low* or vice versa) for both the feedback-based and the prediction-based policy are the same and follow the discussion in Section 3.3. The main difference is that by predicting the arrival of the *High* state the

lack of space.

system can prefetch the user working set before the state changes and avoid performance penalties in a large portion of user requests. The feedback-based policy is a reactive one: it acts *after* it detects a state change. As a result, user requests arriving right after the state change suffer from performance penalty of being served at the slow tier, till the fast tier is warmed up. The larger the fast tier, the longer it takes to warm it up and the higher the performance penalty of the feedback-based policy.

The prediction-based policy is designed to fall-back on the feedback-based policy: if the prediction time for a *High* state is in the future but the *High* state is already detected, then system work is stopped and the fast tier is warmed-up with the working set reactively.

3.4.1.1 Measurement Results

Using `fio`, we generate a random reads workload accessing data stored in our server. The intensity of user IO requests is shown in Figure 3.8 and it emulates very closely the load pattern of user requests shown in Section 3.2. Note that without any system work, the response time of user IO requests remains in the same range of about 150ms. All IOs are served from the fast tier. The user throughput however does increase by one order of magnitude as the arrival intensity increases. This confirms that the storage system does not suffer from queuing delays and it has the capacity to sustain more user load.

We add on the same experiment some system work. Initially, the system work is slowed down as to not interfere with the user workload performance. In Figure 3.9 we show the same user workload interleaved with system work with very low intensity, i.e., the system throughput reaches up to 121 IOPS for the first 100 minutes. In the next 100 minutes the intensity of system work increases and its throughput reaches 5968 IOPS, a two-orders of magnitude increase from the first half of the experiment. User performance is not impacted in the first half of the experiment, but system work here is minimal. The figure plots the

throughput of both user and system work, as well as the response time of the user workload. In the second part of the experiment (100 min to 200 min) where systems work is launched, the increase in user response time is 50-fold, while its throughput is very low.

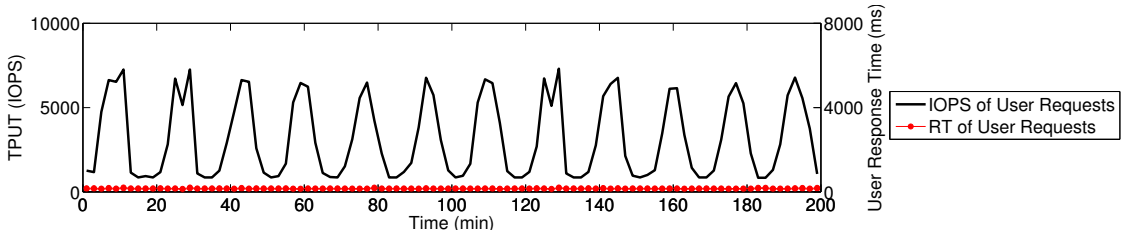


Figure 3.8: User IOPS (throughput) and user response time over time, *user-only* policy.

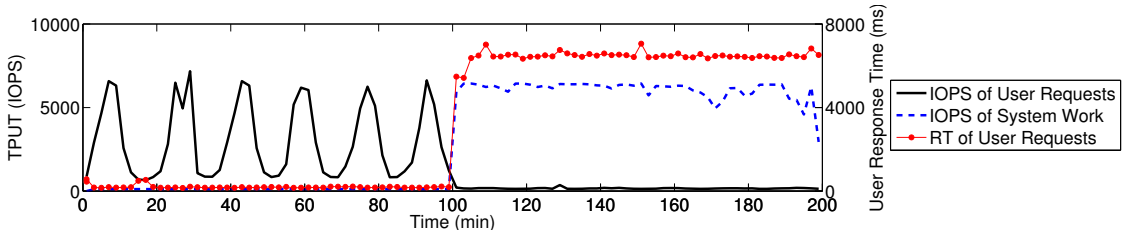


Figure 3.9: User and system IOPS (throughput) and user response time over time. In the first half of the experiment there is minimal systems work, in the second half systems work is increased by two orders of magnitude.

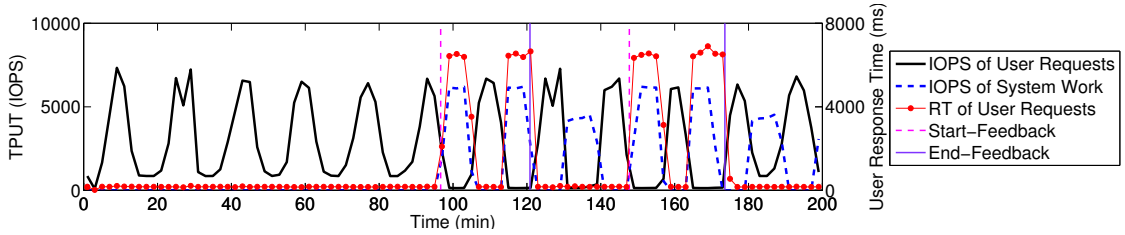


Figure 3.10: User and system IOPS (throughput) and user response time over time. Till the 100th minute, there is only user workload. In the time periods from the 100th to the 120th minute and the 150th to the 170th minute, the feedback policy is launched. The prediction policy is launched from the 120th to the 150th minute, as well as after the 170th minute to the end of the experiment.

In Figure 3.10 we do the same experiment but we now activate the feedback and the prediction-based policies in the second half of the experiment, when the system work is

launched/ The feedback policy was used from the 100th to the 120th minute as well as for the time period between the 150th to the 170th minute. In the rest of the time periods, the prediction-based policy is used. The graph shows that when the prediction policy is activated (time periods: 120-150, and 170-200), user response time remains unscathed, both with respect to throughput and response time. In the time periods when the feedback policy is used, we see high throughput of system work but also user response time that are orders of magnitude greater than the user-only case.

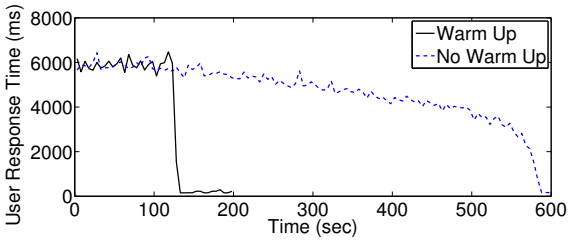


Figure 3.11: Response time with warm up and without warm up across the experiment time.

What makes the difference in user performance between the feedback and prediction-based policies is the timely fast-tier warm up. Figure 3.11 captures this effect. In this experiment, we use a very small data set of 1GB and let the fast tier warm up 1) by the accesses of the regular user workload (i.e., no explicit warm up) and 2) by specifically bringing the working set up to the fast tier (warm up) via a sequential read of the user working set. While it takes only 130 seconds to bring 1GB of data into our fast tier by reading it sequentially (300 seconds and 700 seconds for 4GB and 8 GB of data, respectively), it takes 600 seconds to fully warm up the cache by the user workload alone (more than one hour for the 4GB and 8GB working sets). As the working sets and fast tier capacities grow to TBytes, it becomes imperative not only to warm up the cache before the high user load starts, but doing it proactively (with the aid of our model) than reactively (feedback). A more fine-grained evaluation of our predictive policy is done via trace-driven simulation in

the next subsection.

3.4.2 Simulation Results

In order to evaluate our predictive approach at a fine-grain level and better understand its statistical properties, we experiment also with a trace-driven simulation that allows us to change the various parameters of the experiment. The simulation, in particularly, allows us to analyze the benefits of the predictive methods as the size of the fast tier increases, without been constricted by the specific hardware as in the case of our limited testbed.

Our simulation is driven by the traces described in Section 3.2. Since the traces contain only the arrival process, the service process is assumed to be exponentially distributed with a mean service rate that ensures that the response time remains flat during the full range of user arrival intensities. We simulate a two-tiered storage system with configurable capacities and user active data set sizes to experiment with different fast tier warm up times (i.e., 1 minute, 15 minutes, and 60 minutes).

We have implemented the feedback-based and the prediction-based policies for scheduling system work. As a baseline comparison, we also report performance data when no system work is launched (i.e., we also present the user-only case). The simulation, similar to our measurement experiments, is built such that when the system is experiencing high user arrival intensities, the system work is stopped. When the system experiences low arrival intensities then the system serves both user requests and system work. The differences between the predictive and feedback approaches lies on the exact time *when* the system work is stopped and resumed.

For the results that we present here, the model is already trained with two weeks of trace data and we present results when the model is applied in one of the days (day 20, see Figure 3.2). Figure 3.12 illustrates the points where the two model predictions differ. The

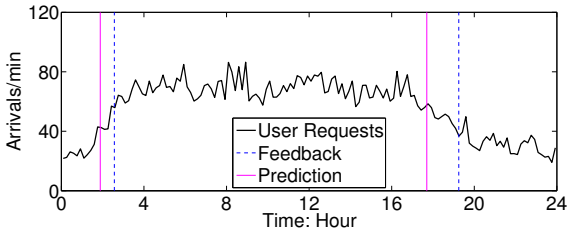
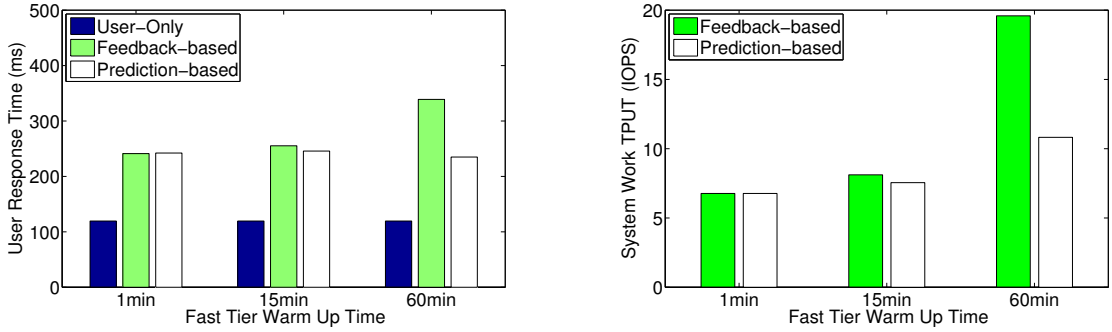


Figure 3.12: Predicted state change by feedback and prediction methods.

lines that are marked as feedback illustrate the points in time where the arrival intensity changes after observation. Note the feedback uses on-line detection, so there is a delay between the true change point and detected moment The prediction lines correspond to the time stamps where the model predicts that there is an imminent load change. Due to the stochasticity of the arrival intensities and the Markovian-based model, the prediction model deviates from the real change that is accurately detected by the reactive, feedback-based method. Yet, this accuracy of the feedback model becomes almost a moot point since it cannot be used to enable tier warm up before the high user load.



(a) average user response time

(b) throughput of system work

Figure 3.13: Performance comparisons via simulation. Note the throughput for system work is null in the user only case.

In our simulation experiments we compare the average user response time and the average system work throughput between different fast tier capacities (measured by the time it

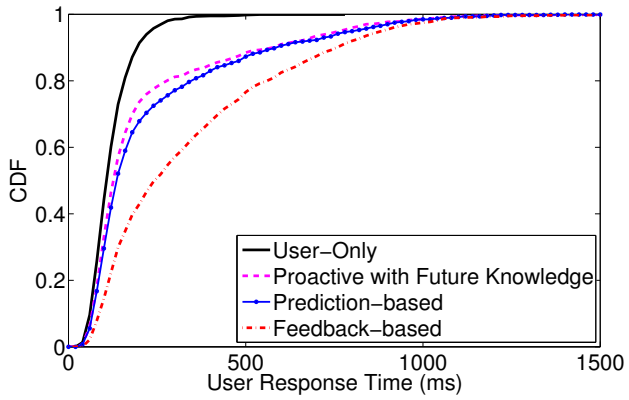


Figure 3.14: CDF of user response time.

takes them to warm up). The expectation is that the predictive method would detect the incoming *High* state and proactively warm-up the fast tier earlier than the feedback method detects the *High* state after the fact. As a result, system work runs for longer stretches under the feedback method than the predictive method. Consequently, the predictive method completes less system work, but also maintains high user performance. Note that the larger the fast tier, the higher the benefits of the predictive approach, otherwise the system is left to operate under high user arrival intensities and a cold fast tier for longer periods of time. These behaviors are captured in Figure 3.13 and further corroborated by Figure 3.14 where the CDF of the user response time is plotted under the scenario of a fast tier requiring 60 minutes to warm up. In the other two cases of smaller fast tiers, the differences between the feedback method and the predictive methods are not as pronounced. As a final note, note that in Figure 3.14 we have also added the ideal proactive policy that assumes full knowledge of the future workload to initiate the tier warm up. The response time CDF of the prediction-based policy is very close to that of the ideal one, which further argues about the effectiveness of the model prediction.

3.5 Chapter Summary

We have examined the effects of various workload interleaving techniques in tiered storage and have demonstrated the performance benefits of a stochastic, Markovian-based model that can be used to first learn and then predict cyclic patterns in user workload intensity. Using a variety of user workload traces for production systems we have demonstrated the robustness of the model as it effectively suggests when to deploy and when to stop the deployment of system storage features in order to better, resulting in serving systems work during the most opportune low utilization periods.

Chapter 4

Fine-granularity Performance Isolation

Different from Chapter 3, in this chapter, we emphasis on providing a performance prediction model, that is able to achieve *finer* control of the resource isolations in nowadays storage systems. Often in such systems, data analytics workloads co-exist with high priority user workloads that operate within strict service level objectives (SLOs). Data analytics workloads, e.g., personalized advertising, sentiment analysis, product recommendation, database replication, dominate many systems today. Different than traditional internal work (e.g., garbage collection, snapshots, upgrades), data analytics work requires faster reaction in order to provide timely information [57, 31, 132], e.g., a delayed advertisement event update could cause reduced income or a product recommendation should occur before the user leaves the site. Since data analytics to enhance user experience and regular user traffic share the same hardware, their effective resource management can greatly enhance business value and user satisfaction.

Scheduling user traffic and data analytics work in the same system is a challenging task. Scheduling data analytics too aggressively may cause user traffic to suffer from SLO

violations. If scheduled too conservatively, data analytics work could not finish in time, thus could loose its value. With user workload traffic that is repeatable and periodic across different time scales [132, 111, 121], it is natural to interleave data analytics work with user workload at periods of low user demands.

Key to the effective deployment of any policy that interleaves data analytics with SLO-bound user level work, is the prediction of fluctuations in the user workload and especially identifying a priori heavy or spiky loads. Here, we propose *NeuQ*, a neural network/queuing hybrid solution: the queueing model that is the basis of co-scheduling decisions is significantly enhanced by neural networks to improve its accuracy. The queueing models that we use predict the magnitude of the potential performance interference of co-scheduling user and data analytics workloads. An important parameter of the queueing model that greatly affects its prediction accuracy is a priori knowledge of the upcoming arrival intensity, this is successfully provided by the neural network.

To illustrate the effectiveness of *NeuQ*, we consider tiered storage systems as a use case. In storage systems flash-based technologies (e.g., DRAM, SSD) are widely integrated into data centers and scaled-out cloud storage systems [89, 62, 1]. Despite their performance advantages over the traditional disk- or tape-based storage technologies (e.g., HDD), their much higher cost per byte prevents flash-based storage technologies to completely replace traditional disk or tape based storage devices. Hybrid, tiered architectures that integrate flash with various disk technologies are common alternatives.

Tiered storage systems adopt a hierarchical design: fast tiers using flash storage devices aiming at boosting performance and slow tiers using HDD devices for the purpose of balancing capacity and cost, as well as for improving reliability [53, 86]. Their efficiency is based on moving data to the right tier based on statistics of data access frequencies. Data moving, i.e., *what* portion of data ought to be transfered and *when* this should take

place, affect greatly performance as access times and capacities across different tiers differ by orders of magnitude. This greatly depends on how the upcoming workload intensity is known in advance and is vital for *NeuQ*'s success in offering co-scheduling decisions, as the effects of tier warming greatly depend on timely information on this measure.

NeuQ guarantees user SLOs while maximizing data analytics throughput. To this end, we do not treat data analytics simply as a best effort workload. Instead, our aim is to schedule it as aggressively as the system allows without violating user SLOs. We stress that the above performance targets are by no means a limitation of the proposed hybrid model. Incorporating different targets (e.g., deadlines) for completion of data-analytics workload are also easily handled, as we show here.

The proposed approach is fully automatic and robust. We validate its correctness and efficiency via trace-driven simulation using two case studies: 1) enterprise traces from Wikipedia [111] and 2) arrival traces in a scaled-out storage back-end of a mid-size web service provider that we have also used in prior work[1]. Our extensive experimental evaluation shows that the prediction of the user traffic arrival intensity and data analytics completion time is remarkably accurate. More importantly, compared to other state-of-the-art scheduling approaches, the proposed solution strictly meets user SLOs while serving aggressively data analytics workloads. In addition, our approach also supports different scheduling objectives.

This chapter is organized as follows. In Section 4.1, we discusses on related work. Section 4.2 presents the hybrid model: the machine learning model for traffic intensity prediction and the queueing model. Section 4.3 presents extensive experimental evaluation via trace driven simulations to verify the correctness and effectiveness of the proposed methodology. We conclude in Section 4.4.

[1]The Wikipedia traces are publicly available [111]. Due of confidentiality agreements, the storage system trace or provider details can not be made publicly available.

4.1 Related Work

Analytical and simulation models have been widely used to quantify the impact of workload changes to application and/or system performance, see [40, 53, 121, 11, 100, 127, 124] and references therein. [121] uses a probabilistic model to define "workload states" via hierarchical clustering. After state discovery, the observed workload is used to parameterize a Markov Modulated Poisson Process that can accurately predict the duration of each state as well as transitions from state to state. ARMA/ARIMA [45] have been adopted in [132] to predict the user traffic overtime in order to achieve cost-efficient capacity planning. However, this prediction method is limited to the linear basis function.

Machine learning techniques are used to overcome the limitations of the linear basis function of ARMA/ARIMA models, and are used for effective characterization of TCP/IP [32] and web server views [65]. Machine learning techniques [33] have been also used for performance prediction of total order broadcast, a key building block for fault-tolerant replicated systems. Ensembles of time series models have been used to project disk utilization trends in a cloud setting [103].

In general, analytical models are restricted by their simplified assumptions while machine learning models are effective in predicting performance for scenarios that have already been observed in the past and fail when new patterns are observed. A gray-box performance model that combines analytical modeling with machine learning has been proposed [38]. The authors advocate the use of analytical models to lower the initial training time of machine-learning based predictors or enhance the accuracy of the analytic model by adjusting its error with the help of machine learning. In contrast to this work, what we propose here is the usage of machine learning to accurately predict specific inputs of a queueing model, which in turn we use to derive scheduling decisions.

4.2 Methodology

In this section, we start with the description of the user workloads used in this work. Then we illustrate how to use a neural network to build a traffic prediction model. Finally, we introduce in details of the *NeuQ* scheduler that is powered by a queueing model.

4.2.1 Workload

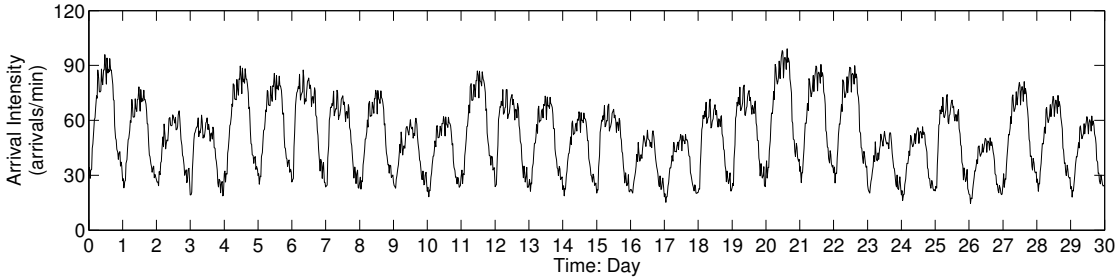


Figure 4.1: User request arrival intensity of storage workload over one month.

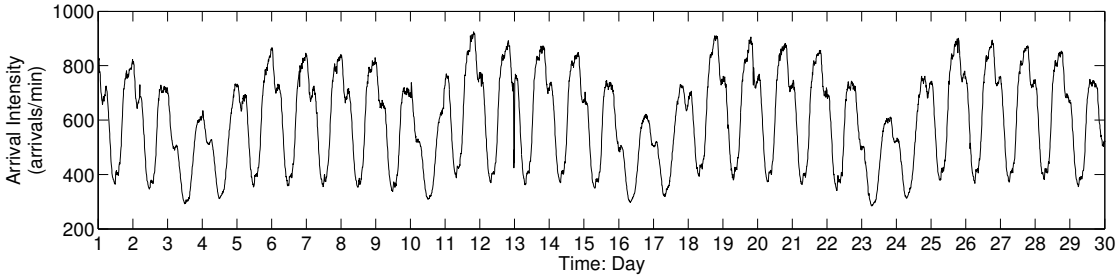
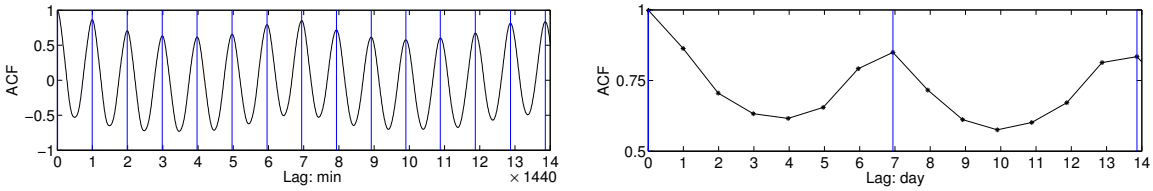


Figure 4.2: User request arrival intensity to Wikipedia over one month in 2007.

Data center workloads often follow periodic patterns across time [17, 11, 121, 14]. In Figure 4.1, we demonstrate the arrival intensity of the storage system workload during one month period [121]. In Figure 4.2, we present the arrival intensity of requests to Wikipedia during October 2007 [111].

Intuitively, the workload of Figure 4.2 shows a distinctive day/night pattern as well as weekday/weekend pattern. To capture this, we carry out some statistical analysis by calculating the autocorrelation of the time series of the arrival process. Autocorrelation is the cross-correlation of a signal with itself [63]. Intuitively, it captures the similarity between observations as a function of the lag between them. Autocorrelation values are in the $[-1, 1]$ range, the closer the lag autocorrelation to 1, the closer the two observations. Zero values indicate no relationship among the observations, while negative values indicate that the range of values of the two observations is diametrically different.

The autocorrelation of user arrival intensity for the Wikipedia workload at varying time lags is shown in Figures 4.3(a) and 4.3(b), that report on autocorrelations at the minute lag (10-minute granularity) and day lag, respectively. The figures verify clear daily and weekly patterns, as also observed in Figure 4.2. Autocorrelation reaches a maximum value per day across all lags illustrating a clear daily pattern, ditto for Figure 4.3(b) that illustrates a clear weekly pattern. Similar autocorrelation patterns are also observed for the storage workload. In the following section, we use these properties to train a neural network that can model user arrival intensity.



(a) Minute granularity (b) Day granularity

Figure 4.3: Autocorrelation of arrivals for different granularities. Note for (a), the points are at 10-minute granularity, the ticks in x-axis is multiplied by 1440, e.g., 2 represents $2 * 1440$ minutes or 2 days, so the lag is up to 14 days.

4.2.2 Neural Network Model

Neural networks are a class of machine learning models that can capture periodicity within different time scales. Traditional time series models such as ARMA/ARIMA [45] and Holt-Winters exponential smoothing [51] are usually limited by the linear basis function. Neural networks can instead model non-linear functions of the input, which makes them effective for time series modeling [41]. A time series analysis consists of two steps: first building a model that represents a time series, and then using the model to forecast future values. If a time series has a regular pattern, then a value of the series should be a function of previous values. An artificial neural network consists of multiple nodes, or *neurons*, which are interconnected to mimic a biological neural network [56]. These neurons have adaptive weights, tuned by a learning algorithm, which enables neural networks to approximate non-linear functions of their inputs. The adaptive weights describe the connection strengths between neurons, activated during training and prediction.

To build a neural network model for a time series, selecting the most *relevant* input that can describe the *trend*, *season*, and *noise* is utterly important. To take care of *trend* and *season*, or in other words, to capture the *long-term* pattern, we make use of the correlogram in Figure 4.3(b). The figure shows that when the lag equals to seven days, the user traffic is highly and positively correlated. Therefore, as input to our traffic model, we choose the user arrival intensities of exactly one week ago. To capture the temporal change or *noise*, which can be seen as a *short-term* pattern, we look into Figure 4.3(a) and see that for lag = 10 min the arrival intensities have the highest correlation value. This suggests to consider the user arrival intensities of 10 min ago as another input to the model. With the above inputs as features and current observations, we can train a neural network. When training a neural network, the data are divided into three distinct subsets [58]: the training set, that is used to train the weights (model parameters) of neural network; the validation set,

which is used to minimize overfitting thus ensure a generalized solution; and the test set, which is used for evaluating the accuracy of the trained neural network model. Here we use the neural network toolbox provided by MATLAB [37] to train our traffic prediction models.

When training a neural network, even with the same data, the trained model can have different accuracy depending on the configured parameters, e.g., different initial weight values, different divisions of data used for training, validation, and test samples. Our primary attempt to avoid using badly-behaved neural networks (that result in poor prediction), was to train several neural networks on the same data set and select the one with the smallest mean squared error. However, since the future largely remains unknown, it is possible that the "best" neural network fails to do a good prediction in the future. To mitigate this potential problem, we use an ensemble method [103], which averages predictions from different neural networks. Even though the ensemble method may not always provide the optimal prediction, it consistently produces accurate predictions without requiring manual checking.

Last but not least, the computational complexity of the neural network training is not significant, as the prediction model can forecast upcoming traffic for up to a week ahead, suggesting that it is sufficient to update the neural network model as often as once per week for the data in hand. The frequency of training can be adjusted based on different needs.

4.2.3 Co-scheduling

As Figure 4.2 shows, user traffic demonstrates peaks and valleys, suggesting that one could aggressively co-schedule data analytics work during the "low" user periods. Our intention is to quantify how much additional work one could co-schedule such that overall system

utilization increases while user SLOs are respected. Because we are aiming to provide a scheduling approach that is easy to deploy and integrate with other management tools, we refrain from making scheduling decisions for user requests too frequently, as this could result in significant overhead. Therefore, our framework divides the time into small windows t_w and makes scheduling decisions only at the beginning of each window[2].

4.2.3.1 Performance model for user traffic in a tiered storage system

To simplify presentation, we assume that the time window that the user SLO is computed is the same as the scheduling window size t_w.

Arrival process: Although in large time windows the arrival process shows a periodical pattern, within each small time window t_w, the arrival process can be viewed as a Poisson process [63].

Service process: In a 2-tiered storage system[3], if the working set of the coming IO request is in the fast-tier (e.g., SSD), the request is served in fast tier. Otherwise, the coming IO request is served in the slow-tier (e.g., disk). Here we assume the service process for each tier follows an exponential distribution. The service process for the 2-tiered system can be described by a hyper-exponential model [93] with mean service time:

$$E[s] = h \times E[s_1] + (1-h) \times E[s_2], \tag{4.1}$$

where $E[s_1]$ and $E[s_2]$ are the mean service times for the fast tier (tier 1) and slow tier (tier 2), respectively. h is the fast tier hit rate defined as the probability that a request is served from the fast tier. The maximum value of the fast tier hit rate h_{max} is determined by the

[2]We assume $t_w = 1$ minute in our experimental evaluation, but this could be adjusted according to the specific system requirement.

[3]For presentation reasons, we use a 2-tiered storage system to explain our methodology, but it could be easily extended to storage systems with more tiers. This discussion also applies to caching.

workload characteristics and the capacity of the fast tier, e.g., if the entire working set can loaded into the fast tier, then the maximum hit rate is 1, in which case all the requests are served in the fast tier. Based on Eq. 4.1, the expectation of the squared service time can be computed as follows [93]:

$$E[s^2] = 2! \times (h \times E[s_1]^2 + (1-h) \times E[s_2]^2), \tag{4.2}$$

where $E[s_1]^2$ and $E[s_2]^2$ are the square of mean service times for the fast tier (tier 1) and slow tier (tier 2), respectively.

Queuing model: Based on the above assumptions, the 2-tired storage system can be modeled by a $M/H_2/1$ queue for each small time window. We use the Pollaczek-Khinchine formula [107] to compute the average response time of an $M/H_2/1$ queue:

$$RT = E[s] + \frac{\lambda \times E[s^2]}{2(1-\rho)}, \tag{4.3}$$

where $E[s]$ is the mean service time, $E[s^2]$ is the mean of squared service time, λ is average arrival intensity, and ρ is the system utilization

$$\rho = \lambda \times E[s]. \tag{4.4}$$

Combining Eqs. 4.1, 4.2, and 4.4 into Eq. 4.3, we have:

$$RT = h \times E[s_1] + (1-h) \times E[s_2] + \frac{\lambda \times (h \times E[s_1]^2 + (1-h) \times E[s_2]^2)}{1 - \lambda \times (h \times E[s_1] + (1-h) \times E[s_2])}. \tag{4.5}$$

From Eq. 4.5, it is clear that within each time window t_w, the average user response time RT is a function of the average arrival intensity of user requests λ, the mean service time for each tier ($E[s_1]$ and $E[s_2]$), and the fast tier hit rate h.

4.2.3.2 NeuQ Scheduler

Co-scheduling data analytics work may have a performance impact on the user traffic. From our performance model above, the first three parameters (λ, $E[s_1]$, and $E[s_2]$) only depend on the characteristics of user workload and performance of hardware devices, so co-scheduling has no impact on these parameters. However, co-scheduling data analytics work does impact the fast tier hit rate h because it evicts the user working set from the fast tier and reduces the probability of serving user requests in the fast tier. Therefore, in order to meet the user SLO (RT_{target}), the fast tier hit rate of user traffic needs to be maintained above a threshold that is computed from the user SLO and arrival intensity. We compute the threshold (h_{target}) based on Eq. 4.5 by representing the fast tier hit rate as a function of the user response time target and arrival intensity:

$$h_{target} = \frac{\lambda \times (RT_{target} + E[s_1] - E[s_2]) - \sqrt{P}}{2\lambda \times (E[s_1] - E[s_2])}, \tag{4.6}$$

where:

$$\begin{aligned} P = {} & \lambda^2 \times (RT_{target}^2 + 2RT_{target} \times E[s_1] + 2RT_{target} \times E[s_2] + (E[s_1] - E[s_2])^2) \\ & - 2\lambda \times (RT_{target} - E[s_1] - E[s_2]) + 1. \end{aligned} \tag{4.7}$$

Note that the fast tier hit rate depends on both user response time target and arrival intensity because the latency is composed of service time and queuing waiting time, e.g., during periods of high arrival intensity, a higher fast tier hit rate is needed to achieve the same response time target than during low arrival intensity periods.

Now the question becomes how to make scheduling decisions such that the data analytics work can be completed as fast as possible without affecting the user SLO. Because the data analytics work is usually measured by the average completion time of submitted jobs or the throughput, it is important to quantify and maximize the (cumulative) time

slot allocated to the analytics work within each time window t_w. When the system is not as busy with user requests, there are 2 choices: (i) scheduling the data analytics work for time t_{DA}, which reduces the fast tier hit rate or (ii) explicit warming up the fast tier, which recovers the fast tier hit rate.

In order to maximize the time slot allocated to data analytics work but maintain the fast tier hit rate above the threshold h_{target}, different scheduling choices need to be made based on the arrival intensity in the near future. Recall that the future arrival intensity can be predicted using the neural network model introduced in Section 4.2.2. Therefore, we can estimate the fast tier hit rate h_{DA} after scheduling for t_{DA} time units data analytics work:

$$h_{DA} = h_{begin} - \frac{t_{DA}}{t_{evict}}, \tag{4.8}$$

where h_{begin} is the fast tier hit rate at the beginning of a time window and t_{evict} is the time to evict the entire user working set from a fully warmed up fast tier. Since the data analytics work is very intensive, t_{evict} can be approximated by the time to explicitly warm up the fast tier from completely cold to fully warmed up, which we define as $t_{warmup1}$, this can be easily obtained by a quick profiling experiment such as the one shown in Figure 3.11. Assuming that the remaining of idle time is used for explicit warm up of the fast tier, the fast tier hit rate h_{warmup} is:

$$h_{warmup} = min\{h_{begin} + \frac{(1 - \lambda \times E[s]) \times t_w - t_{DA}}{t_{warmup1}}, h_{max}\}, \tag{4.9}$$

recall that h_{max} is the maximum value of the fast tier hit rate. In addition, serving user requests also changes the fast tier hit rate (no explicit warm up):

$$h_{user} = h_{begin} + \frac{\lambda \times E[s] \times t_w}{t_{warmup2}}, \tag{4.10}$$

where $t_{warmup2}$ is the time to non-explicitly warm up the fast tier from completely cold to fully warmed up, which can be easily obtained by a quick profiling experiment. Combining Eqs. 4.8, 4.9, and 4.10, we have the fast tier hit rate at the end of the time window h_{end}:

$$h_{end} = min\{h_{begin} - \frac{t_{DA}}{t_{warmup1}} + \frac{(1 - \lambda \times E[s]) \times t_w - t_{DA}}{t_{warmup1}} + \frac{\lambda \times E[s] \times t_w}{t_{warmup2}}, h_{max}\}. \quad (4.11)$$

Since each time window is very small, we assume that the fast tier hit rate only changes at the end of the window, but stays the same within the window. To meet the SLO, the fast tier hit rate needs to be maintained at or above the threshold i.e., $h_{end} \geq h_{target}$, therefore

$$t_{DA} \leq ((h_{begin} - h_{target}) \times t_{warmup1} \times t_{warmup2} + (1 - \lambda \times E[s]) \times t_w \times t_{warmup2} + \lambda \times E[s] \times t_w \times t_{warmup1}) \times \frac{1}{2 \times t_{warmup2}}. \quad (4.12)$$

From the above inequality, we can quantify the maximum amount of time to be allocated to data analytics work without violating user SLOs. The co-scheduling decisions are based on Eq. 4.12. We stress that this inequality is *critical* for the success of workload co-scheduling as it regulates the amount of data analytics work that the system can systain in order to not violate the user SLO.

Because the targeted fast tier hit rate changes with the arrival intensity, scheduling decisions need to be evaluated well in advance. Here we determine how early a scheduling decision needs to be evaluated such that there is enough time to fully warm up the fast tier during this period. We define $t_{advance}$ as:

$$t_{advance} = (h_{max} - h_{current}) \times t_{warmup1}, \quad (4.13)$$

where $h_{current}$ is the current fast tier hit rate. Based on the predicted arrival intensity after $t_{advance}$, a correct scheduling decision (the time allocated to data analytics work) for the current time window can be made.

In addition, based on the time slot allocated to the data analytics work (Eq. 4.12), we can estimate the throughput of data analytics work $Throughput_{DA}$ as follows:

$$Throughput_{DA} = \frac{t_{DA}}{s_{DA}} \tag{4.14}$$

where s_{DA} is the average service time of data analytics work.

If the scheduling target is meeting deadlines for data analytics work, then the throughput of data analytics work needs to meet the above requirements. Assume that the throughput requirement is $Throughput_{target}$, then we have the following:

$$Throughput_{DA} = \frac{t_{DA}}{s_{DA}} \geq Throughput_{target}, \tag{4.15}$$

therefore,

$$t_{DA} \geq Throughput_{target} * s_{DA}. \tag{4.16}$$

The above inequality indicates the minimum amount of time to be allocated to the data analytics work in order to meet the deadlines.

4.2.3.3 NeuQ Scheduler with Capacity Planning

The interesting question is whether it is possible to meet both user response time SLO RT_{target} and data analytics work throughput SLO $Throughput_{target}$ as in practice, each workload usually has its own performance target. Such scheduling target is achievable through capacity planning, thanks to the elastic storage techniques [69]. When the size of fast-tier increases, serving data analytics work has less impact on evicting user working set that resides in fast-tier, so increasing the capacity of fast-tier allows scheduling more aggressively data analytics work during high user traffic intensity periods. Let's denote v_{evict} as the speed of user working set being evicted from the fast tier. v_{evict} can be computed as:

$v_{evict} = \frac{\delta_h}{t_f}$, where δ_h is the change of user work hit rate during the data analytics work serving time period t_f. Recall t_{evict} is the time to evict the entire user working set from a fully warmed up fast-tier (with hit rate of $h_{upper} = 1$) to completely cold (with hit rate of $h_{lower} = 0$), so we have: $v_{evict} = \frac{h_{upper} - h_{lower}}{t_{evict}} = \frac{1}{t_{evict}}$. Let's denote fast-tier capacity as C, then the time to evict all data from fast-tier t_{all} can be computed as: $t_{all} = \frac{C}{\mu_c}$, where μ_c is the service rate of the fast tier. Since the time to evict all data is equal to the time to evict user working set from fully cached to empty, we have:

$$t_{evict} = \frac{C}{\mu_c}. \tag{4.17}$$

For each time window:

$$h_{end} = h_{begin} + \frac{\rho * t_w}{t_{warmup_1}} + \frac{(1-\rho) * t_w - t_{DA}}{t_{warmup_2}} - \frac{t_{DA}}{t_{evict}}. \tag{4.18}$$

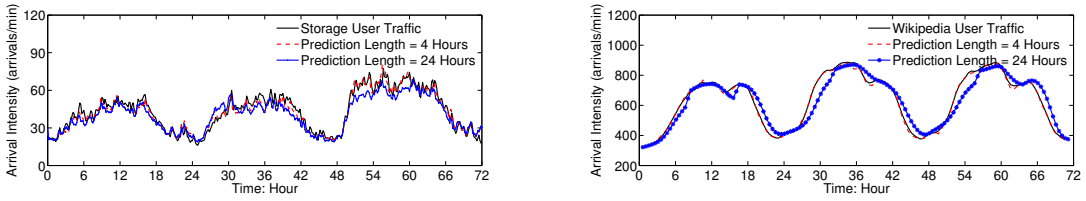
Together with Eq. 4.12, Eq. 4.16, Eq. 4.17, and Eq. 4.18, it is possible to compute the minimum fast-tier capacity to achieve the SLOs of both user work and data analytics work.

4.3 Performance Evaluation

An integral part of the effectiveness of workload co-scheduling is the ability to predict accurately the upcoming user workload. We first evaluate the accuracy of the prediction model and then compare the performance of the proposed approach with other methods in the literature. We also show several scenarios that illustrate the effectiveness of *NeuQ*. Finally, we demonstrate *NeuQ* can support different scheduling targets.

4.3.1 Traffic Prediction

We drive our simulations using the storage workload in [121] and the Wikipedia trace that is shown in Figure 4.2 (both workloads with granularity of 1 minute). We trained neural networks for the two workloads and used the trained models to predict incoming traffic for the time period of the next three upcoming days. We also illustrate the model with two different prediction lengths, i.e., 4 hours and 24 hours. Here, if the prediction length equals to K hours, the neural network directly predicts the arrival intensities in the next K hours. In the proposed *NeuQ* scheduler, the traffic intensity information is usually needed only a few hours ahead, so such prediction length can fully satisfy the scheduler's needs. Consistent with the workload analysis of Section 4.2.1, for each observation during training, we select the observations from the most recent time window and the one from one week ago as features. In Figure 4.4, we show the traffic predictions with two prediction lengths, as well as the actual traffic. The figure illustrates that for both of the storage and Wikipedia traces, the shorter the prediction length, the more accurate the predictions. Yet, even predicting 24 hours ahead, the overall prediction accuracy is still good.



(a) Storage Trace (b) Wikipedia Trace

Figure 4.4: Traffic predictions from neural networks with different prediction lengths.

To quantify better the quality of the two prediction lengths, we use the *absolute percentage error* (APE) metric which is defined as:

$$APE = \frac{|Prediction - Actual|}{Actual} \times 100\%. \tag{4.19}$$

When APE is close to 0, it suggests an accurate prediction. Figure 4.5 illustrates the CDFs of the absolute percentage error for the two traffic predictions. For the storage trace, Figure 4.5(a) shows that the 80 percentile of the APE for the 4 and 24-hour predictions, is less than 12 and 18 percent respectively. For the Wikipedia trace, Figure 4.5(b) shows that for the 4-hour prediction length, almost all the APEs are no more than 10 percent, i.e., predictions are very accurate. For the 24-hour case, over 70 percent of the APEs are less than 10 percent, and nearly 100 percent of them are no more than 20 percent.

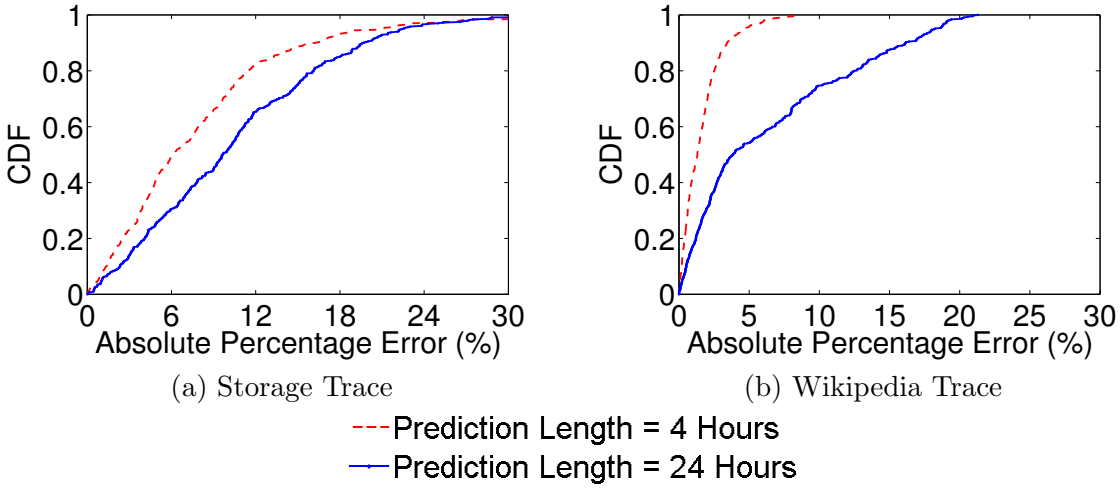


Figure 4.5: CDF of absolute percentage error of the neural network predictions.

4.3.2 Scheduler Comparisons

Here, we compare our *NeuQ* scheduler with two other state-of-the-art scheduling approaches in the literature. We also compare to a scheduling approach that only uses neural network traffic prediction without any help from queueing models.

- *On-line Feedback*

  The on-line feedback method [35] collects measurements during the most recent time

window to check whether user performance requirements are met or violated. If met and provided that the system is in a low utilization state in the current window, then data analytics work is added. In the interest of showing the maximum benefit of this scheduler, we assume that we know all future upcoming workload, i.e., we are certain about the length of low/high utilization times due to user workload *a priori*.

- *State Prediction*

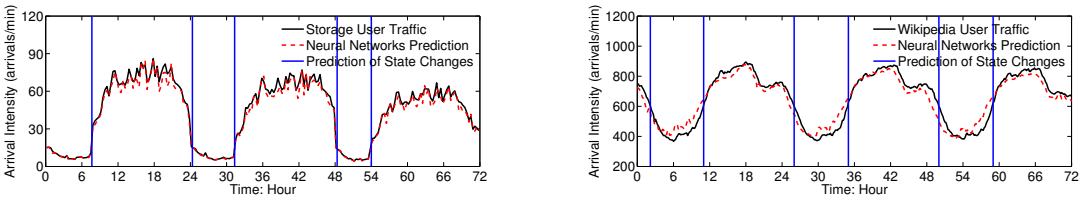
  The state change prediction based method [121] divides the traffic into high/low states, based on the average arrival intensity. It makes use of a Markovian Modulated Poisson Process (MMPP) model to predict when the high state starts and ends. Based on the state prediction, this method only schedules the data analytics work in the low utilization state, while pro-actively warming up the fast tier with the active user working set right before the arrival of a high utilization state.

- *NeuralNet Scheduler*

  The NeuralNet scheduler decides the amount of data analytics work to schedule based on the user arrival intensity prediction. Here the amount of data analytics work to schedule in each time window increases/decreases linearly with the predicted user arrival intensity in that time window. For example, if for the current window w_1, λ_1 is the average user arrival intensity, and the amount of scheduled data analytics work is n_1, then for the incoming time window w_2 with predicted average user arrival intensity equal to λ_2, the NeuralNet scheduler schedules $\frac{\lambda_1}{\lambda_2} \times n_1$ data analytics work in w_2. Intuitively, if the user arrival intensity is predicted to increase, then the NeuralNet scheduler schedules less data analytics work in the incoming time window. Otherwise, more data analytics work is scheduled.

- *NeuQ Scheduler*

The *NeuQ* scheduler makes scheduling decisions (how much and when to schedule data analytics work) based on the hybrid model developed in Section 4.2.



(a) Storage Trace (b) Wikipedia Trace

Figure 4.6: Storage system and Wikipedia user traffic for simulation, with the state changes and traffic predictions.

We conduct trace-driven simulations on the storage workload in [121] and the Wikipedia user traffic (days 26, 27, and 28 in Figure 4.2). Because the *NeuQ* scheduler and the state prediction based scheduler depend on the accuracy of their workload model, we illustrate in Figure 4.6 how well the MMPP model predicts changes in system state as well as the neural network prediction, both methods are quite effective.

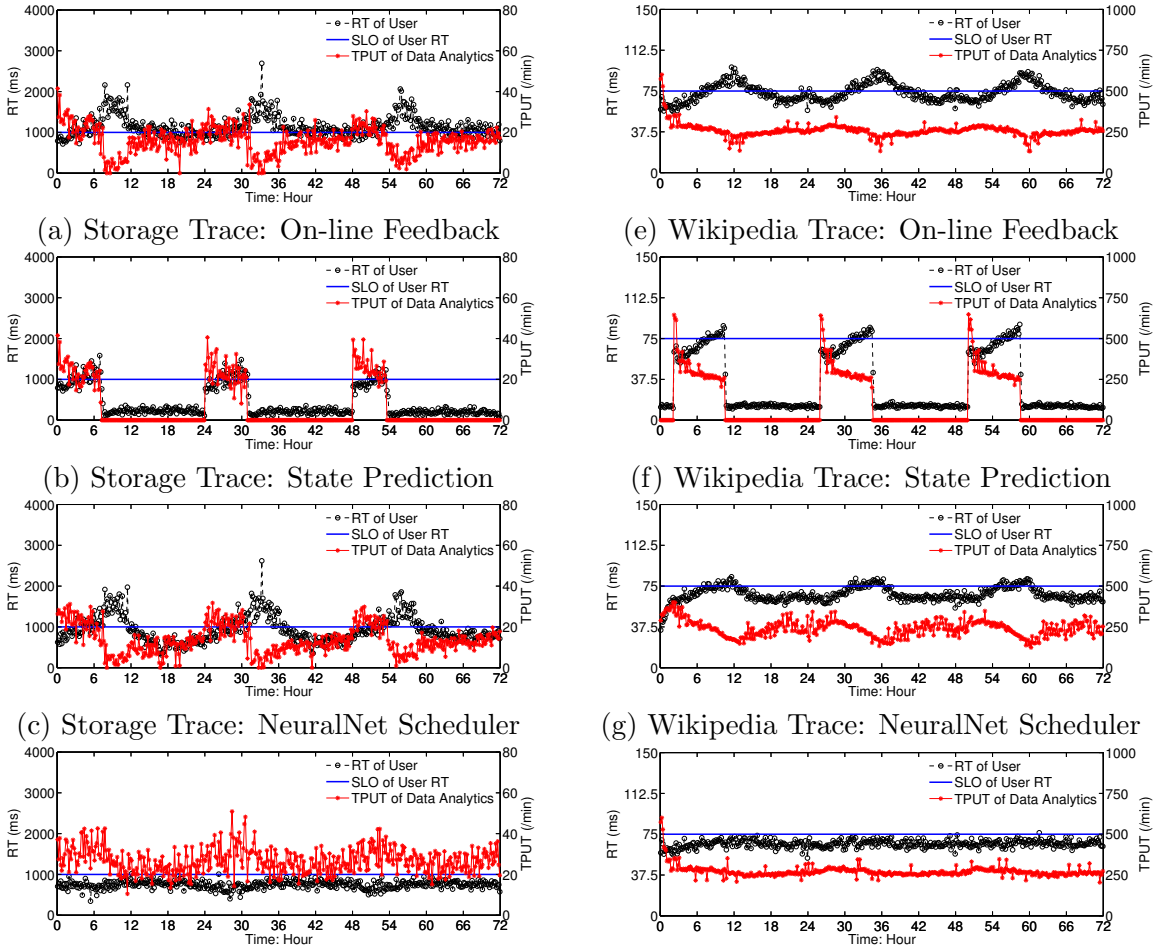
We start by comparing how fast data analytics work each method can complete, given an SLO for user performance. We assume that the service times for the fast and slow tiers are exponential distributed, and that the average service time between the two tiers differs by two orders of magnitude. For the data analytics work, we assume that the average time to finish one unit of work equals to the mean slow tier service time and we use throughput to measure how fast it can be scheduled. Figure 4.7 illustrates the user RTs and data analytics throughputs for the four policies. In each graph, the horizontal lines represent the pre-defined user SLOs. Figure 4.7(a) shows that during heavy user traffic periods (e.g., the 6th, 30th and 54th hour), user SLOs are consistently violated due to the aggressive scheduling of data analytics. Similar phenomena are observed in Figure 4.7(e). The system recovers after the violation is detected. The MMPP-based state prediction method, see Figures 4.7(b) and 4.7(f), is more conservative and refrains from scheduling

data analytics work while the system is in high utilization. Since this scheduler pro-actively warms up the fast tier, it contains SLO violations. However, the stochastic nature of the underlying Markovian-based model results in unavoidable inaccuracies for the *exact* time that the system changes, which results in SLO violations when the high state approaches. In Figures 4.7(c) and 4.7(g), the NeuralNet scheduler is evaluated. The figures show that although less data analytics work is scheduled in heavy user traffic periods than during hours of low traffic, the user SLO is still violated. Figures 4.7(d) and 4.7(h) illustrate the performance of *NeuQ*. With *NeuQ*, there are no SLO violations while data analytics work is served aggressively, and with even higher throughput as with on-line feedback. We also list the percentage of user SLO violations, throughput of data analytics, and its coefficient of variation (CV) for the above four policies in TABLE 4.1. *NeuQ* achieves 2× to 3× higher data analytics throughput without any user SLO violation, while for the other policies, the percentage of user SLO violations is much higher and still less efficient on data analytics throughput.

Table 4.1: Average performance analysis for simulations

	Storage Trace		
	% User SLO Violations	$TPUT_{DA}$ *(/min)*	*CV of* $TPUT_{DA}$
On-line Feedback	68.29	18.24	0.42
State Prediction	11.11	7.58	1.71
NeuralNet Scheduler	32.41	15.53	0.52
NeuQ Scheduler	0.00	26.45	0.23
	Wikipedia Trace		
	% User SLO Violations	$TPUT_{DA}$ *(/min)*	*CV of* $TPUT_{DA}$
On-line Feedback	40.05	257.38	0.16
State Prediction	11.81	110.99	1.45
NeuralNet Scheduler	15.51	240.71	0.21
NeuQ Scheduler	0.00	271.69	0.13

To further illustrate the advantages of our proposed method, we present the CCDFs that illustrate the tails of average user response times of each time window in Figure 4.8.



(a) Storage Trace: On-line Feedback

(e) Wikipedia Trace: On-line Feedback

(b) Storage Trace: State Prediction

(f) Wikipedia Trace: State Prediction

(c) Storage Trace: NeuralNet Scheduler

(g) Wikipedia Trace: NeuralNet Scheduler

(d) Storage Trace: *NeuQ* Scheduler

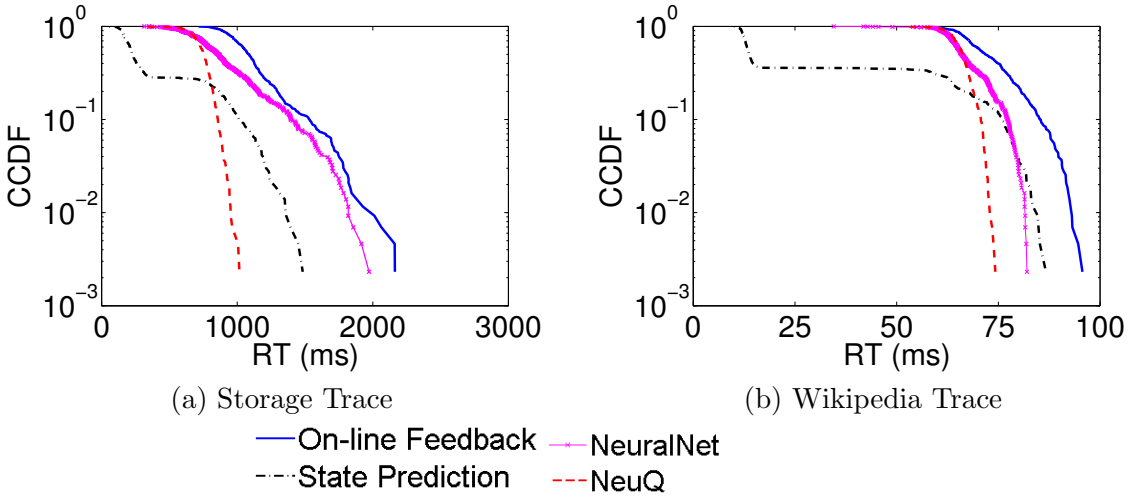
(h) Wikipedia Trace: *NeuQ* Scheduler

Figure 4.7: Performance comparisons via simulation.

NeuQ manages to strictly respect the user SLO (1000ms for the storage trace and 75ms for the Wikipedia trace).

4.3.3 Model Effectiveness

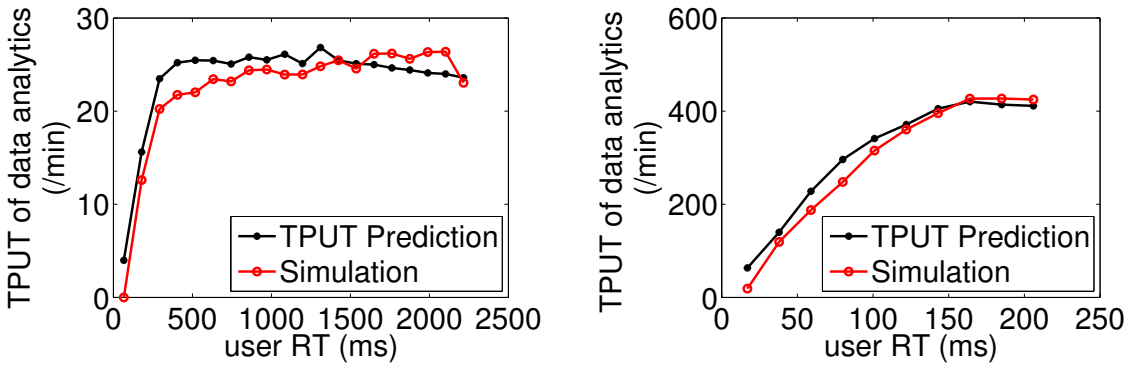
The performance prediction model that is developed in Section 4.2 is the core of the proposed *NeuQ* scheduler. Recall that the model allows us to regulate the amount of data analytics work to be co-scheduled such that a certain SLO is met for the user work.



(a) Storage Trace (b) Wikipedia Trace

Figure 4.8: CCDF of average user response time.

Similarly, if one needs to increase the throughput of data analytics, this would result in affecting the user RT as well.



(a) Storage Trace (b) Wikipedia Trace

Figure 4.9: Throughput of data analytics versus user response time: model (Eq. 4.14) and simulation (*NeuQ* scheduler).

Figure 4.9 illustrates this relationship between user RT (x-axis) and data analytics throughput (y-axis) for the storage and Wikipedia workloads. The figure plots the relationship of these measures as obtained both by simulation and by using the prediction

model of Section 4.2 (Eq. 4.14). Both model and simulation numbers are in good agreement, well-capturing the relationship trends of both measures. Further, by using Figure 4.9 one could estimate the maximum data analytics throughput to be achieved given a certain SLO or conversely the sustained average user RT if a certain throughput for data analytics work is expected.

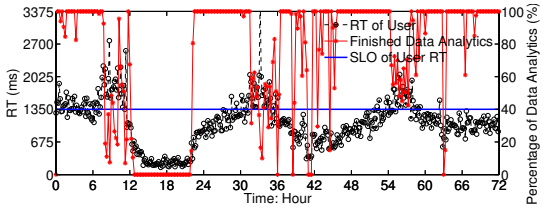
4.3.4 Different Scheduling Targets

NeuQ can be used to support different scheduling targets. We first demonstrate the scheduling target scenario of finishing the data analytics work by the pre-defined deadlines while preserving the performance of user workload as much as possible. We use Storage trace for evaluation and show two different scheduling targets:

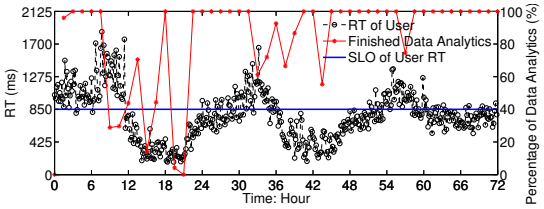
- *Scheduling Target 1*: the deadline of data analytics work is 15 minutes for 500 units of work. The SLO of user workload is SLO 1350ms.
- *Scheduling Target 2*: the deadline of data analytics work is 1.5 hours for 1500 units of work. The SLO of user workload is SLO 850ms.

Scheduling Target 1 has a tighter deadline for data analytics work while relatively loose SLO for the user workload, and Scheduling Target 2 is on the contrary. Both scheduling scenarios consider meeting the deadline of data analytics work as first priority and meeting SLO of user workload as a secondary target. The data analytics work that is not finished by the deadline is dropped so that its impact is not propagated to the future analytics work.

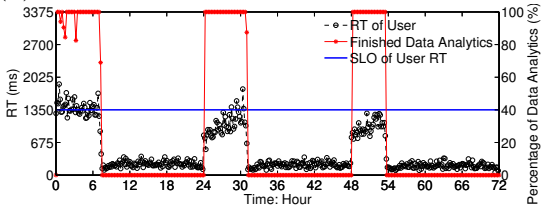
The results are presented in Figure 4.10. The left y-axis is the user response time measured in ms and right y-axis is the percentage of the finished data analytics work. The x-axis represents the elapsed time (3-day period). For Target 1 (left column of graphs), the



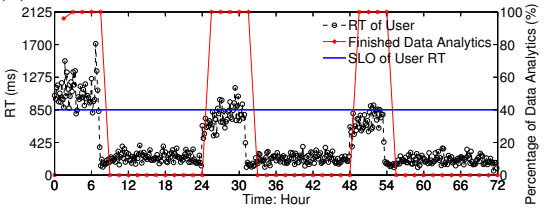
(a) Scheduling Target 1: On-line Feedback



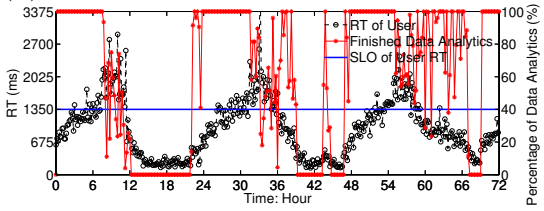
(e) Scheduling Target 2: On-line Feedback



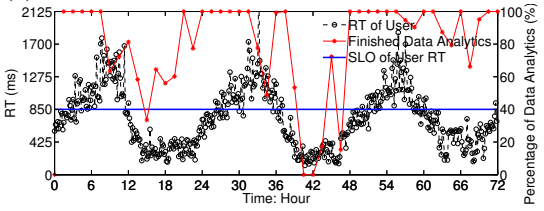
(b) Scheduling Target 1: State Prediction



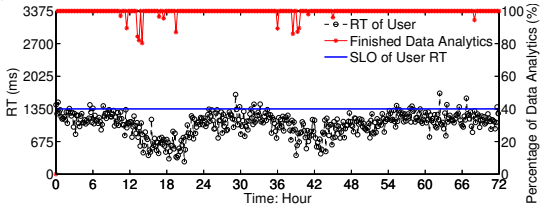
(f) Scheduling Target 2: State Prediction



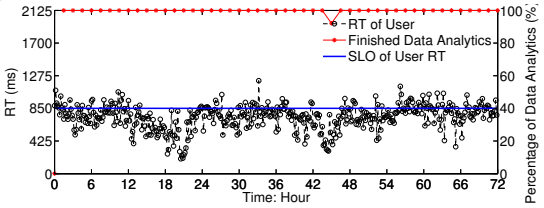
(c) Scheduling Target 1: NeuralNet Scheduler



(g) Scheduling Target 2: NeuralNet Scheduler



(d) Scheduling Target 1: *NeuQ* Scheduler



(h) Scheduling Target 2: *NeuQ* Scheduler

Figure 4.10: Performance comparisons via simulation. Scheduling Target 1 (left column of graph): the deadline of data analytics work is 15 minutes for 500 units of work and the user workload SLO is 1350ms. Scheduling Target 2 (right column of graph): the deadline of data analytics work is 1.5 hours for 1500 units of work and the user workload SLO is 850ms.

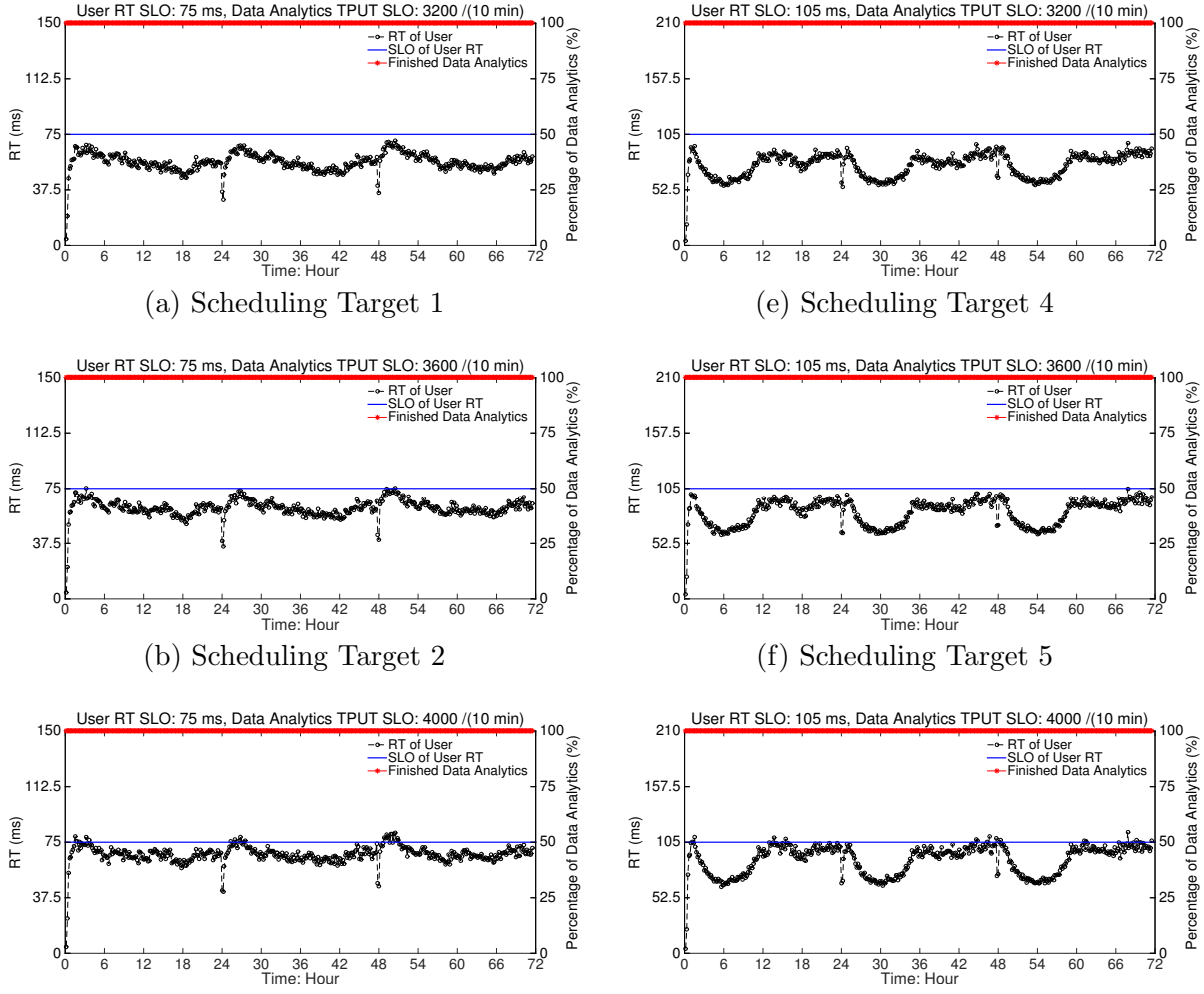
results suggest that only *NeuQ* can meet the deadlines of data analytics work (there are a few exceptions, but very few and all above 80%) and all other methods fail. Meanwhile, *NeuQ* also consistently archives the SLO of user workload, this is not the case for other methods. Target 2 (right column of graphs) shares the same requirement for meeting deadlines of data analytics work. Given the stricter SLO, none of the methods can meet the

deadlines of data analytics work while also achieving the SLO for user workload. However, it is clear that *NeuQ* results in smallest violation of the SLO of user workload among these methods. The above experiments suggest that *NeuQ* has great use potential in reaching other targets. For the experiments presented in Figure 4.10, especially for Scheduling Target 1, it is necessary to tolerate high user SLOs to meet the 15 minutes deadline.

Then we evaluate the scheduling target scenario of meeting both user response time target and data analytics work throughput target through capacity planning. We use Wikipedia trace for evaluation and show six different scheduling targets:

- *Scheduling Target 1*: the deadline of data analytics work is 10 minutes for 3200 units of work. The SLO of user workload is SLO 75ms.
- *Scheduling Target 2*: the deadline of data analytics work is 10 minutes for 3600 units of work. The SLO of user workload is SLO 75ms.
- *Scheduling Target 3*: the deadline of data analytics work is 10 minutes for 4000 units of work. The SLO of user workload is SLO 75ms.
- *Scheduling Target 4*: the deadline of data analytics work is 10 minutes for 3200 units of work. The SLO of user workload is SLO 105ms.
- *Scheduling Target 5*: the deadline of data analytics work is 10 minutes for 3600 units of work. The SLO of user workload is SLO 105ms.
- *Scheduling Target 6*: the deadline of data analytics work is 10 minutes for 4000 units of work. The SLO of user workload is SLO 105ms.

These scheduling targets represent a variety selection of different user workload SLO and data analytics work SLO combinations. The results are presented in Figure 4.11. The left y-axis is the user response time measured in ms and right y-axis is the percentage of the



(a) Scheduling Target 1

(e) Scheduling Target 4

(b) Scheduling Target 2

(f) Scheduling Target 5

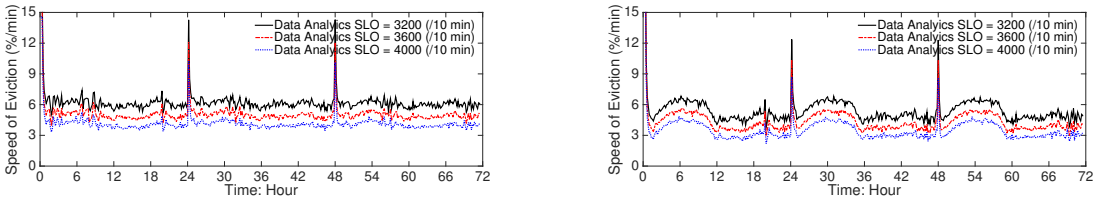
(c) Scheduling Target 3

(g) Scheduling Target 6

Figure 4.11: User response time and finished data analytics work under various scheduling targets.

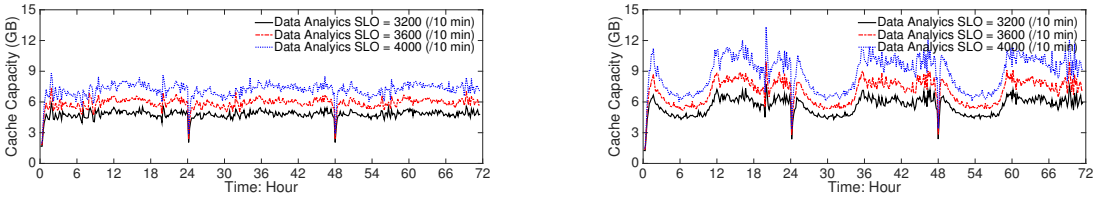
finished data analytics work. The x-axis represents the elapsed time in 3-day period. Due to the interest of space, the results of other methods are not shown here. Because other methods do not take into consideration of both user workload SLO and data analytics work SLO, as expected, the SLOs could not be met at the same time. For *NeuQ*, it consistently archives the SLOs of both user workload and data analytics work in different scenarios, see Figure 4.11. We also plot the user working set evicting speed from fast-tier and fast-tier

capacity demands overtime in Figure 4.12 and Figure 4.13 respectively. These plots show how *NeuQ* correctly estimates the user working set evicting speed and fast-tier capacity demands under different system load so that to plan in advance the fast-tier capacity to accommodate both user workload and data analytics work in each time window.



(a) User response time SLO of 75 ms (b) User response time SLO of 105 ms

Figure 4.12: User working set evicting speed from fast-tier overtime with different scheduling targets.



(a) User response time SLO of 75 ms (b) User response time SLO of 105 ms

Figure 4.13: Fast-tier capacity demands overtime with different scheduling targets.

4.4 Chapter Summary

Co-scheduling data analytics workloads with user workloads in multi-tiered storage systems is a challenging problem. In this chapter, we propose *NeuQ* scheduler, a hybrid co-scheduling approach using machine learning and queueing models, that applies neural networks to predict user workload intensities and then appropriately adjusts the input to a queueing model in order to consistently meet user SLOs. Trace-driven simulations show

that *NeuQ* can effectively reach performance targets under different user workloads and different performance/scheduling targets from commercial systems.

Chapter 5

Prediction of Data Center Time Series

Previous Chapter introduces a vanilla neural networks based time series prediction method for system workloads. However, this method is limited by the its input (all the past observations) into the neural networks and its operational cost. As a result, in this chapter, we focus on providing a selection method for the input data into the neural networks, for the time series prediction in cloud data centers. Effective workload characterization and prediction hold the answers to the conundrum of efficient resource allocation in distributed and scaled out systems. Being able to *accurately* predict the upcoming workload within the next time frame (i.e., in the next 10 minutes, half hour, hour, or even week) allows the system to make proactive decisions, rather than reactive ones. Proactive decisions can be used with superior performance in storage systems by timely warming up the cache with the working set [121, 128], especially in systems where traditional internal work (e.g., garbage collection, snapshots, upgrades) is interleaved with the user workload during opportune times. Proactive scheduling of data analytics work can result in personalized advertising, sentiment analysis, or timely product recommendation, i.e., before the user leaves the

site [57, 31, 132].

Virtual machine (VM) consolidation and migration is another example where accurate prediction of the physical machine utilizations can guide effective system usage [30, 83, 116]. In all of the above cases, prediction of the intensity of peak loads and of their timings becomes key to the effective launching of proactive management.

To maintain performance at tails, e.g., at high percentiles of response times, resource management policies [57, 30, 116], need to address the demands of peak loads instead of average loads only. Depending on the capability of predicting peak load magnitudes and timings, resources can be multiplexed at various degrees across users and across time. Such predictions can guide VM consolidation in data centers.

In this chapter, we focus on data center workloads within the private cloud operated by IBM and used by major corporations for their IT needs. Prior work on workload characterization at IBM data centers [16] focused on statistical analysis of the usage of specific components of the virtual and physical machines, e.g., CPU and IO [16, 14]. This statistical analysis focused on averages, percentiles, and trends, aiming to a better understanding of how the workload evolves across a two-year period, but largely ignored the time series of the various performance metrics. In this chapter, we focus on these time series and develop methodologies for accurate prediction of various workload metrics and especially peaks and their timings.

Classic time series models such as ARIMA [45] can be used for online prediction. Such models first need to be trained using past observations and can predict the upcoming workload. Alternatively, neural networks can be used in the same manner and provide a black box approach to predict the future, especially to predict events that have been observed in the past. Superior to the classic time series models that use a linear basis function, neural networks model input using non-linear functions, which improves their

ability to handle more complex observations. Features gathered from observations are not all equally informative; some are relevant, while others are noise. Key to effective neural network prediction is the discovery of the appropriate features. Neural network training is then conducted based on these.

In this chapter, we develop a robust framework for prediction of data center time series (PRACTISE) and illustrate the flexibility of such a black box approach by showing remarkable accuracy in usage prediction of data center workloads in the wild. We focus on four components: CPU, memory, disk, and network. We focus on an actual production workload and particularly on 56 physical machines that host 775 virtual machines during a time period of 61 days. Based on observations of the workload pattern and its periodicity, we extract the features that identify the time periods in which the repetitive patterns occur. We also develop a bagging module [20] and an online updating module to improve the stability, accuracy, and speed of PRACTISE. We provide detailed comparisons with ARIMA and show that the proposed black box approach offers a significant improvement in predicting resource usage, by reducing average errors by three times. PRACTISE slashes the false negative prediction rates of peak loads to less than 12% and achieves two fold to nine fold improvements in the accuracy of timing predictions. PRACTISE is lightweight and achieves training and prediction by an order of magnitude faster than other methods, which allows it to be used online.

This chapter is organized as follows. Section 5.1 discusses related work. Section 5.2 presents an overview of the workload. Section 5.3 presents the machine learning model. Section 5.4 presents extensive experimental evaluation. Section 5.5 discusses potential use scenarios. We conclude in Section 5.6.

5.1 Related Work

ARMA/ARIMA [45] have been widely used for time series prediction in several systems areas. Tran and Reed use ARIMA to improve block prefetching for scientific applications [109]. They use ARIMA to predict the temporal access pattern and Markov models to identify spatial access patterns and manage to identify what and how much to prefetch. Their predictor is implemented on the Linux file system. In [132] ARIMA is used for effective user traffic prediction for capacity planning. The authors focus on cost-efficient database replication that is driven by the anticipated user traffic within the LinkedIn social network. ARIMA models have also been used in sensor networks to reduce the frequency of sampling and to improve on energy efficiency by transmitting only deviations from the ARIMA-predicted values [66]. Anomaly detection is yet another area where ARIMA models have been used [130].

Machine learning techniques are used to overcome the limitation of the linear basis function of ARIMA models and are used for effective characterization of TCP/IP [32] and web server views [65]. Neural networks are used for performance prediction of the total order broadcast, which is a key building block for fault-tolerant replicated systems [33]. Ensembles of time neural network models have been used to project disk utilization trends in a cloud setting [103]. Neural networks and hidden Markov models are used for automatic IO pattern classification and are evaluated with both sequential and parallel benchmarks [75]. Probabilistic models that define workload states via Markov Modulated Poisson Processes have been used in [121] to interleave workloads with different performance objectives. Machine learning techniques have been widely used for workload prediction [9, 95, 34, 59]. In contrast to these works, we rely on the autocorrelation and automate the entire learning process.

The effectiveness of the proposed neural network approach that we advocate in this

chapter is based on statistical analysis of the workload so that the most relevant features are selected for the training data set. Training the model with careful feature selection significantly improves its accuracy and stability but also increases the speed of training and prediction, making it appropriate to use for online performance prediction and capacity planning. In addition, due to the appropriate feature selection, PRACTISE can provide short-term (e.g., 15 minutes) to long-term (e.g., one day or one week ahead) predictions and achieve excellent accuracy. These superior predictions facilitate robust long-term capacity planning and resource allocation.

5.2 VM Workloads in a Private Cloud

The target systems of this study are IBM private data centers, which are geographically distributed across all continents. These systems are used by various industries, including banking, pharmaceutical, IT, consulting, and retail, and are based on various UNIX-like operating systems, i.e., AIX, HP-UX, Linux, and Solaris. Those systems are highly virtualized, meaning that multiple virtual machines (VMs) are consolidated on a single physical box. Both VMs and boxes are very heterogenous in terms of resource configuration. The average virtualization level per box is ten [16]. We have collected resource utilization statistics from several thousands of VMs and boxes since February 2013. The finest observation granularity is 15 minutes[1]. The analysis here is based on two-month data from March 1, 2013 to April 30, 2013.

We focus on usage of four types of resources: CPU, memory, disk, and network. Using the base observation window of 15 minutes, we collect the following statistics:

- **CPU utilization**: the percentage of time the CPU is active over the observation

[1]Collection of data is done by another IBM branch, therefore, we do not have any control on obtaining data at lower granularity.

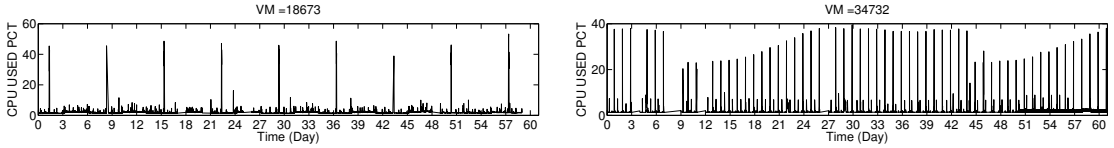


Figure 5.1: CPU utilization over time for two different VMs.

window.

- **Memory utilization**: the percentage of memory capacity used.
- **Disk space usage**: the percentage of allocated disk space used.
- **Network bandwidth usage**: the total network traffic rate measured in mega bits per second (Mbps).

The collected trace data is retrieved via `vmstat`, `iostat`, and supervisor specific monitoring tools.

The VM workloads within the IBM private cloud exhibit clear periodic patterns over time [16], see Figure 5.1. The figure focuses on two different VMs and illustrates the CPU utilization within successive time windows of 15-minute across all 61 days. The upper plot shows a regular periodic pattern with a period of 7 days, while the bottom one shows a more complex pattern with clear trend changes. We also observe that similar periodic patterns exist in different resources, i.e., memory, disk, and network. To the interest of space, we do not present these results here. Pattern periodicity suggests that there exist opportunities for workload prediction.

To capture and quantify such periodic patterns, we perform statistical analysis of the workloads by computing the autocorrelation of the time series of CPU utilization. Autocorrelation is a mathematical representation of the degree of similarity in a time series and a lagged version of itself. As such, it is ideal for discovering repeating patterns by quantifying the relationship between different points of a time series as a function of the

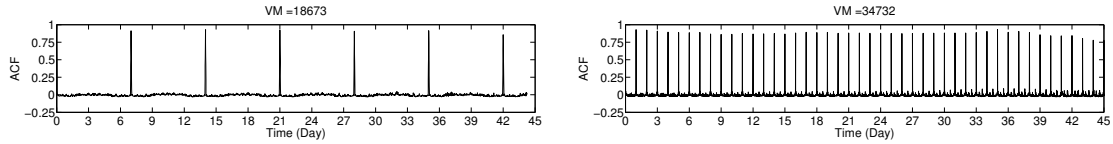


Figure 5.2: Autocorrelation of CPU utilization for the two VMs of Figure 5.1.

time lag [63]. The autocorrelation metric is in the range of $[-1, 1]$. Higher positive values indicate that the two points between the computed lag distance are "similar", i.e., have stronger correlation. Zero values suggest no periodicity. Negative values show that the two points lag elements apart are diametrically different. We show the autocorrelation of CPU resource usage for the two selected VMs in Figure 5.2. It is clear that the autocorrelation becomes high at certain lag values and that the lag values[2] can be different for different VMs. In the following section, we demonstrate how to utilize autocorrelation to select the appropriate features in order to train a neural network that can model the workloads accurately.

5.3 Methodology

A time series prediction model uses past observations to forecast future values. There are different ways to build the time series prediction model. Traditional time series e.g., the ARMA/ARIMA [45] and Holt-Winters exponential smoothing [51] are based on a linear basis function, and as a result they are not effective in predicting complex behaviors. In addition, these models are backward looking only methods, which makes it difficult to capture any new patterns that have not appeared before. Furthermore, the underlying approximation function usually lacks intuitive explanations. For all of the above reasons, it is difficult to improve the prediction accuracy of such types of models. On the other hand, neural networks are capable of modeling input as non-linear functions, which offers great

[2]Note that a lag of 1 corresponds to two intervals 15 minutes apart.

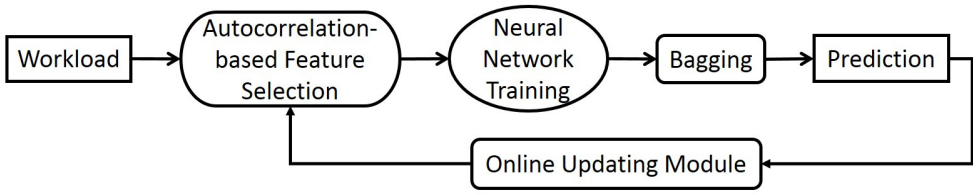


Figure 5.3: Overview of PRACTISE.

potential in handling complex time series [41]. We start from the standard universal neural network toolbox provided by MATLAB [37] and then introduce PRACTISE by selecting more appropriate features, using bagging and online updating to improve its accuracy and stability. An overview of PRACTISE is shown in Figure 5.3. The workload is fed to the autocorrelation-based feature selection module. The selected features then become inputs to the neural network training component. The bagging module processes the aggregated results. Finally, the online updating model monitors the prediction error and triggers a retraining if large errors are detected. In the following, we introduce each component in detail.

5.3.1 Universal Neural Network

Artificial neural networks are inspired by biological neural networks [56] and are composed of many interconnected *neurons*. The weights associated with the neurons are used to approximate non-linear functions of the inputs and are tuned during a training process. Discovering appropriate features is the key to building an accurate neural network model. The universal neural network toolbox provided by MATLAB uses a generalized algorithm for feature selection. To train a neural network, the input data set is usually divided into three subsets [58]: training, validation, and test. The neural network uses the training set to tune its weights and utilizes the validation set to determine the convergence point and prevent overfitting. The test set is used for evaluation of the training accuracy.

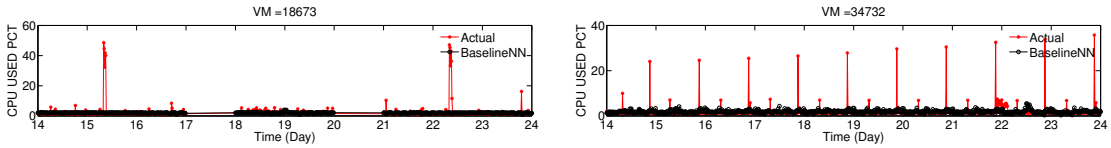


Figure 5.4: CPU workload prediction by the neural network toolbox provided by MATLAB for two different VMs. The two gaps in the first plot are due to the VM being switched off.

To understand the prediction accuracy of the standard neural network toolbox provided by MATLAB, we conducted extensive experiments. Figure 5.4 illustrates the default MATLAB prediction (tagged BaselineNN) of the utilization of the two VMs shown in Figure 5.1. We have trained and validated the neural network using the first 14 days, and we show here the results for days 15 to 24. The figure clearly shows the neural network's pitfalls as prediction accuracy is often poor. Using the standard MATLAB toolbox, the underlying feature selection algorithm is not tuned to optimize the information provided by the repeating patterns. Therefore, we are motivated to explore a better feature selection algorithm for selecting the appropriate features for the data center workloads that we have in hand.

5.3.2 Autocorrelation-based Features

Intuitively, appropriate features should reliably capture periodic behavior, changing trends, and repeating patterns. To identify the appropriate features, we resort to the correlogram in Figure 5.2 because autocorrelation can provide quantitative and qualitative information on the above factors. Figure 5.2 shows that there can be several lags with high positive autocorrelation values. This indicates that there exist several good candidate features that represent short-term to long-term correlation patterns. To automate the process, we use a local maximum detection function to identify the peak points in autocorrelations and use the respective lag values as features for neural network training. In this way, different

correlation ranges from short-term to long-term can all be captured, which improves the effectiveness of the neural network. The remaining steps are the same as with the universal neural network toolbox provided by MATLAB. We stress that the feature selection process is fully automatic.

5.3.3 Bagging

The training features are not the sole factor in the prediction accuracy of a neural network model; the quality of the trained model also depends on other factors. The training data sets are another crucial factor [20]. As discussed earlier, the training set is split into training, validation, and test subsets. Different ways of splitting may result in different samples being used at different stages and therefore result in different trained models. In order to minimize the artificial effects caused by a certain splitting rule, we split the data set randomly several times (e.g., 20 times), and each split trains a different model. In other words, we train a group of neutral network models by using the same data set but with different splits. Each model has its own prediction result. The prediction results from different models together become a distribution of prediction results. To compute the final prediction results from the distribution of prediction results, we first use the 3-sigma rule [110] (e.g., 99% confidence interval) and z-score [77] (e.g., within [-0.85,0.85]) to filter out outliers and then compute the average of the remaining data as the final prediction. Bagging may not always guarantee that optimal prediction is achieved, but it consistently improves prediction accuracy compared to using only a single trained model.

5.3.4 Online Updating Module

In a real cloud environment, there can be sudden or permanent workload changes caused by unexpected events. As neural network models rely on past information to forecast the

Table 5.1: Training time using 14 days' data and prediction length of 1 day.

VM ID	Training Time (sec)		Prediction Time (sec)	
	BaselineNN	PRACTISE	BaselineNN	PRACTISE
18673	300	30	257	10
34732	480	50	360	15

future, workload characterization changes may not be timely reflected in the prediction results. To ensure an agile response to workload characterization changes, we add an online updating module. The update is triggered based on monitoring the prediction errors periodically. If errors suddenly surge, a workload change is suspected and the neural network model is retrained. We emphasize that the computational cost of the neural network training and prediction is not significant thanks to the simple yet efficient feature selection process. Thus, it allows us to retrain the model online quickly at low cost. We demonstrate two examples in Table 5.1 to show how long PRACTISE takes for training and prediction compared to BaselineNN on a machine with 2.8 GHz Intel Core i7 CPU, 16 GB memory and 750 GB SSD. From the table, it is clear that both the training time and prediction time of PRACTISE are very low and that PRACTISE is one order of magnitude faster than BaselineNN. This difference may appear modest, but if prediction has to be done simultaneously for thousands of VMs and multiple resources, it becomes significant. The training and prediction times change linearly with the amount of training data and prediction length.

5.4 Experimental Evaluation

We describe the methods used for the evaluation below:

- **ARIMA**: the standard ARIMA algorithm, used as baseline comparison.

- **BaselineNN**: the default setting of the neural networking toolbox provided by MATLAB, used as a second baseline comparison.

- **PRACTISE**: the workload prediction framework proposed in this chapter.

Evaluation strategies. We use the first 14 days as training data and use the following 46 days as the data for evaluation of the prediction accuracy. With the online updating module, PRACTISE automatically triggers a retraining if the monitored error is outside the confidence interval determined by the 3 sigma rule [110]. We present the results for prediction length of 1 day ahead. We also present two more cases, one predicting 2 hours ahead (short window) and one predicting 1 week ahead (long window). We evaluate PRACTISE by comparing it to ARIMA and BaselineNN in two prediction scenarios: state predictions and timing, and quantified predictions.

State predictions and timing. Several scheduling and management frameworks [121, 75] do not require quantified prediction. Instead, workloads are classified into states, e.g., peak states with relatively high resource usage and non-peak states with relatively low resource usage[3], and only qualitative predictions are needed. In such a scenario, the quality of the prediction is measured by whether the future state can be predicted correctly. In addition, it is critical to be able to not only predict a peak state, but also the time *when* this state occurs. We quantify the timing of the predictions across all VMs in a cumulative way, i.e., we count for how many 15-minute intervals the timing of the prediction of the peak state is delayed.

With the given granularity of 15 minutes, every entry in the traces represents a peak or non-peak state. The threshold between peak and non-peak states is determined via K-means clustering. Because timely predicting peak states is utterly important, we first

[3]Here we focus on peaks because of literature [76, 22], but it can be applied to other utilization levels, not only peaks.

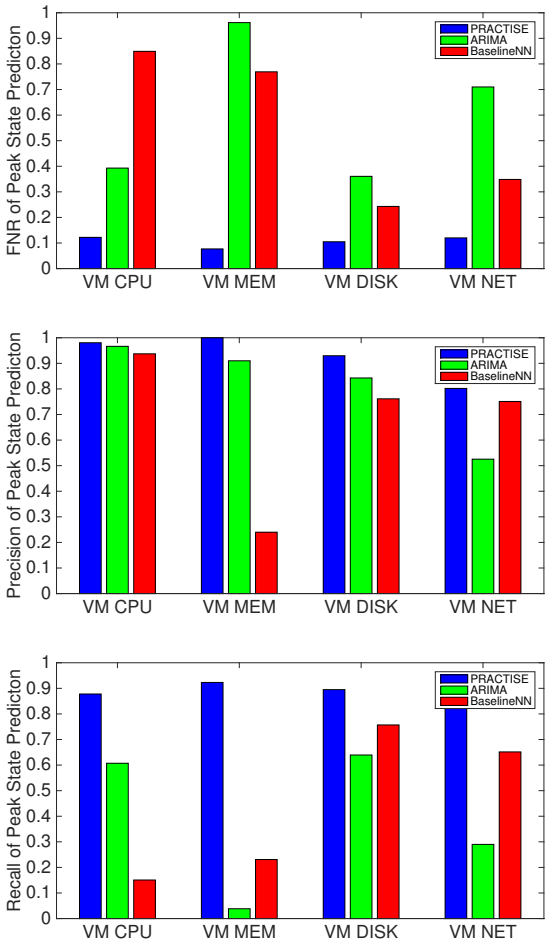


Figure 5.5: Prediction accuracy for peak states. The top plot is the false negative rate (FNR) of peak state prediction, the middle plot is the precision of peak state prediction, and the bottom plot is the recall of the peak state prediction.

evaluate the accuracy of peak state prediction. The top plot in Figure 5.5 illustrates the rate of false negative peak state predictions, which is defined as the number of wrong peak predictions (i.e., states that are predicted as non-peak) divided by the total number of actual peak predictions. Results are across all VMs. PRACTISE consistently achieves less than 12% false negatives across all resources. The false negative rates of ARIMA and BaselineNN are much higher and very random across resources. We also provide two other

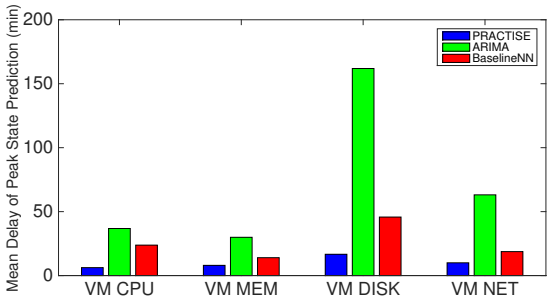
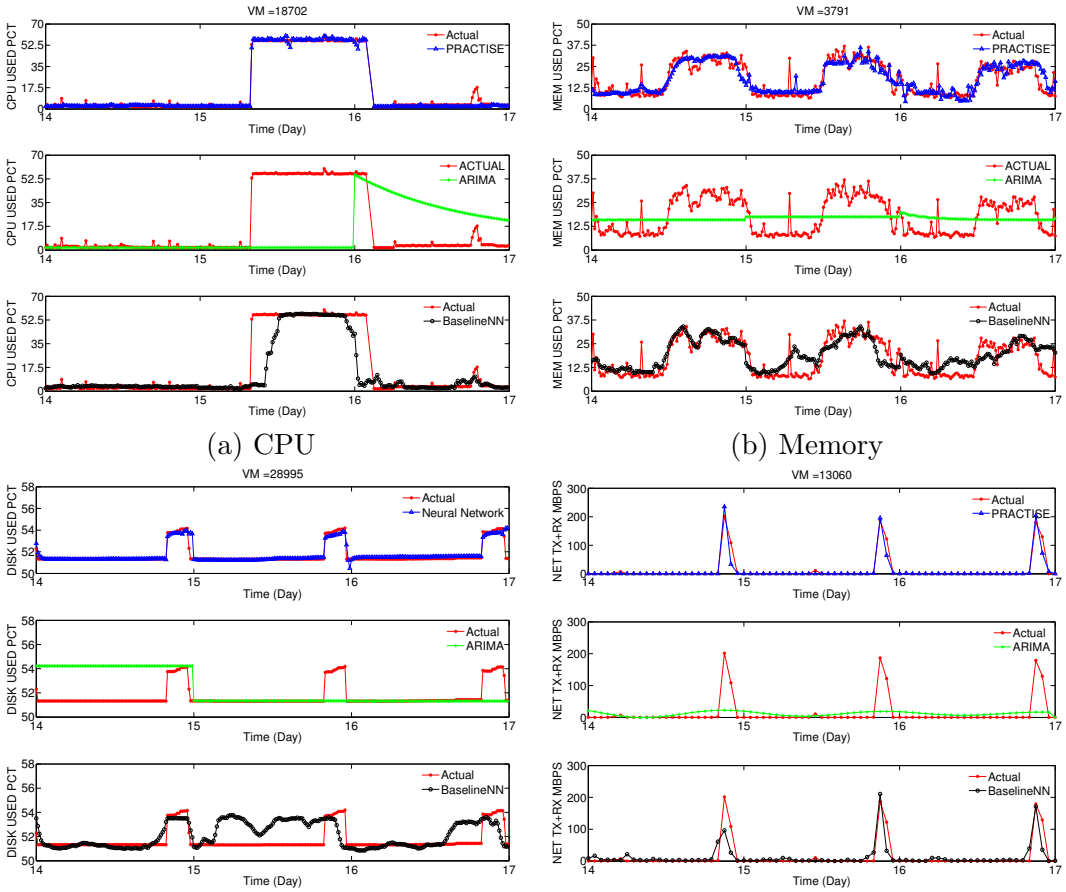


Figure 5.6: Mean delay between prediction and actual occurrence of peak states.

commonly used metrics for evaluating prediction accuracy: precision, which is defined as the fraction of retrieved instances that are relevant and recall, which is defined as the fraction of relevant instances that are retrieved, see the middle and bottom plots in Figure 5.5 respectively. PRACTISE again consistently outperforms ARIMA and BaselineNN with respect to the two metrics, which further validates the accuracy of PRACTISE.

Figure 5.6 illustrates the average delay (in minutes) between the prediction and the actual occurrence of peak states across all 775 VMs. PRACTISE shows a remarkable accuracy across all resources with values dramatically outperforming all other methods, i.e., for VM CPU utilization, the average delay reduces from 36.80 minutes (ARIMA) and 23.82 minutes (BaselineNN) to 6.22 minutes; for VM memory utilization, from 29.93 minutes (ARIMA) and 14.00 minutes (BaselineNN) to 8.00 minutes; for VM disk space usage, from 161.88 minutes (ARIMA) and 45.75 minutes (BaselineNN) to 17.02 minutes; and, for VM network bandwidth usage, from 63.12 minutes (ARIMA) and 18.75 minutes (BaselineNN) to 10.05 minutes.

Quantified predictions. Scheduling and management frameworks do require quantified prediction [131, 117], especially for systems that need to meet certain service level objectives. We first show overtime plots for the actual workload and predicted results. Results for VM CPU utilization, VM memory utilization, VM disk space usage, and VM



(a) CPU (b) Memory

(a) Disk Space (b) Network Bandwidth

Figure 5.7: Prediction for different resource usages of VMs

network bandwidth usage are shown in Figure 5.7(a)-(d) respectively[4]. Each point on the graphs corresponds to the finest workload granularity that is available, i.e., 15 minutes. It is clear that the prediction error of PRACTISE is consistently lower than both ARIMA and BaselineNN, especially for sudden workload surges, thanks to the more appropriate feature selection and bagging used in PRACTISE. Note that predicting sudden workload surges is very important as it can drive timely scheduling/management.

[4]We zoom in a 3-day period for clearer presentation.

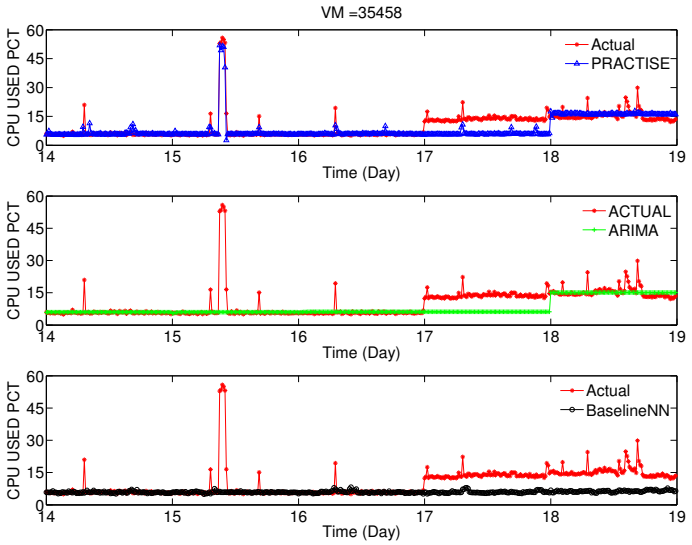


Figure 5.8: Prediction for CPU utilization; the trends changes after day 17.

While the results presented before show cases of consistent periodicity across time, in Figure 5.8 we show a more challenging case where the trends of the periodical pattern change. The results show that PRACTISE can effectively capture this thanks to the online updating component. ARIMA can also react to the trend change, but it fails to capture most peak states. However, BaselineNN can only predict events that have been observed before, and are therefore unable to capture such trend changes. The experiments cover a variety of data center configurations and applications with various usage patterns, but due to the interest of space, we skip other results here.

To quantify the prediction errors, we define the prediction error (APE) as $\frac{|prediction-actual|}{actual}$. We show the CDFs for one specific VM (first row of Figure 5.9) for CPU, memory, disk, and network. We also show the mean and 90th percentile in the legend. The results illustrate clearly that PRACTISE is consistently superior in accuracy compared to ARIMA and BaselineNN as PRACTISE achieves up to 3 times better prediction accuracy than in terms of average prediction errors. The second row of Figure 5.9 illustrates the same

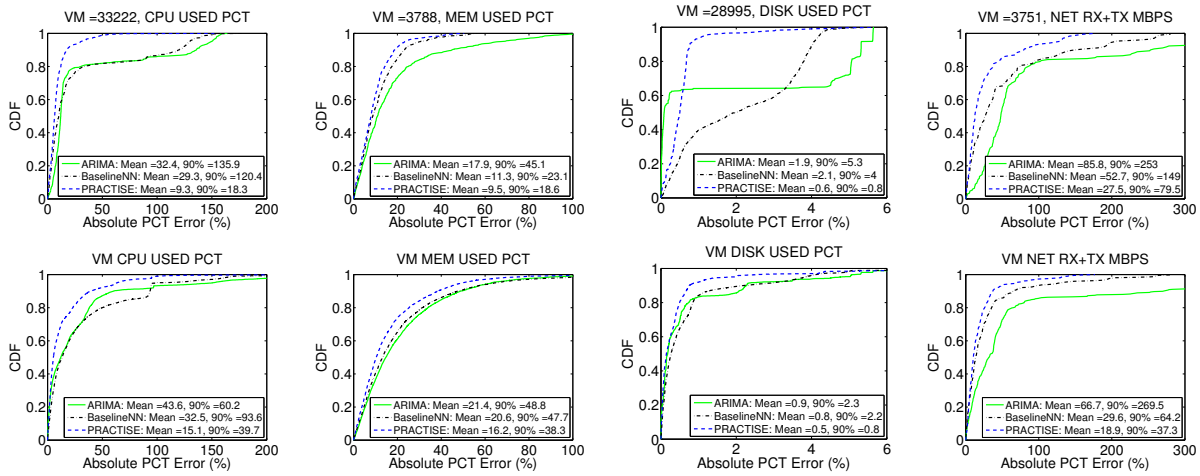


Figure 5.9: Prediction error comparison for different prediction methods. The graphs are for VM CPU utilization (column 1), VM memory utilization (column 2), VM disk space usage (column 3), VM network bandwidth prediction (column 4). The first column is for a selected VM and the second column shows accumulated results over all VMs. Prediction length is 1 day ahead.

information but cumulative across *all* 775 VMs. The superiority of PRACTISE is also clear in this comparison.

To demonstrate the importance of the bagging and the online updating modules, we also compare the prediction errors when bagging and online updating are activated, see Figure 5.10. The results indicate that with these two components, the prediction accuracy can be significantly further improved, which verifies that these two enhancements are non-trivial and useful.

Finally, we evaluate PRACTISE for different prediction lengths. We demonstrate the prediction length of 2 hours and 1 week in Figure 5.11. The results clearly illustrate that PRACTISE consistently outperforms ARIMA and BaselineNN for different prediction lengths. In addition, the change in the size of the prediction window does not affect robustness.

Challenging cases. PRACTISE relies on the autocorrelation values as features for neural network training. Here we evaluate a challenging case with poor autocorrelation

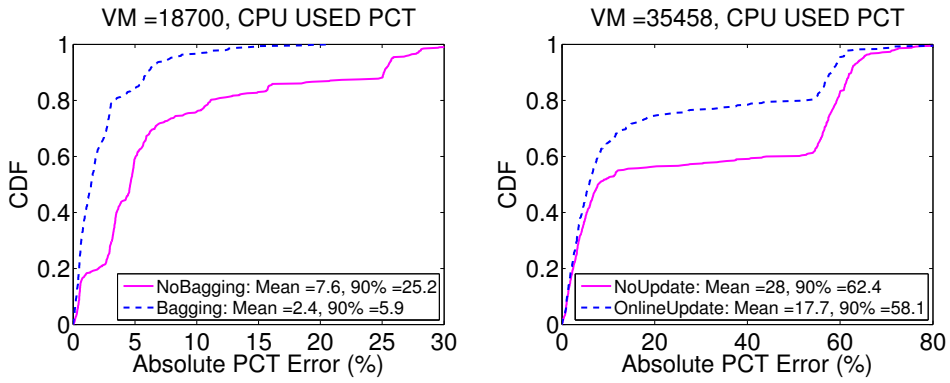


Figure 5.10: Prediction error comparison of VM CPU utilization with and without bagging (top plot), and with and without online updating module (bottom plot).

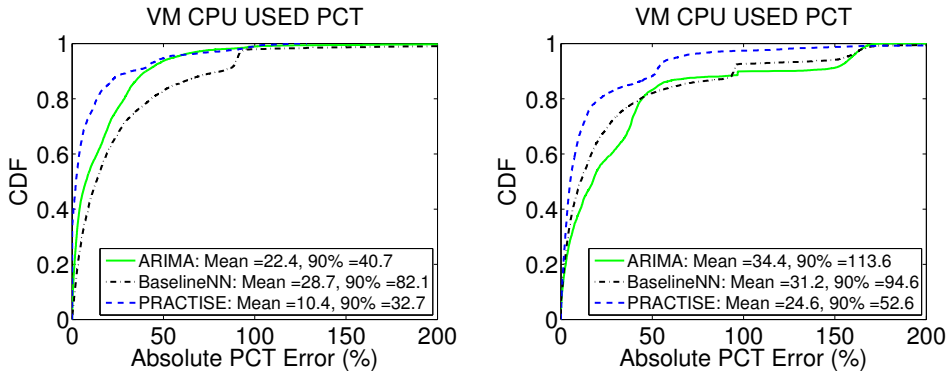


Figure 5.11: Prediction error comparison of VM CPU utilization for prediction length of 2 hours (top plot) and 1 week (bottom plot). The results are accumulated across all VMs.

structure, see the top plot of Figure 5.12. The figure illustrates the autocorrelation plot of the memory utilization of a VM. The autocorrelation function switches between positive and negative values, which makes feature selection very challenging. The CDF of prediction errors for this VM is presented in the bottom plot of Figure 5.12. The results suggest that even for this case, PRACTISE still achieves relatively good prediction accuracy and clearly outperforms ARIMA and BaselineNN. The robustness in prediction is due to the fact that PRACTISE does not solely rely on the autocorrelation features but also on bagging and online updating which contribute to more sophisticated predictions. We conclude that

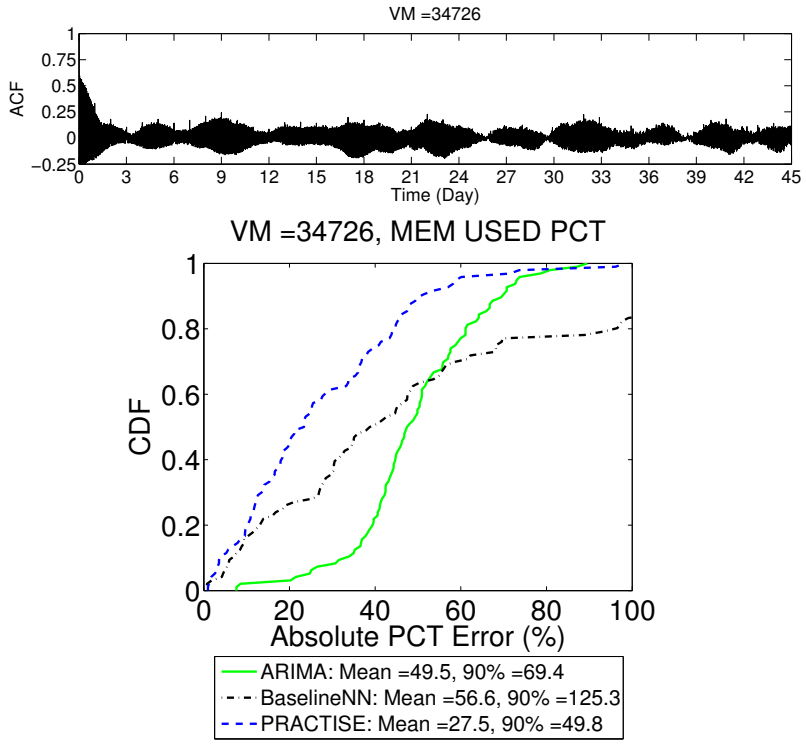


Figure 5.12: A challenging case (VM 34726). Autocorrelation (top plot) and prediction error comparison (bottom plot) of memory utilization for different prediction methods.

PRACTISE is effective, efficient, and robust.

5.5 Discussion

In this chapter, we provided the first important step that is required for VM consolidation and/or load balancing: a framework for efficient and accurate prediction of future load, and in particular peak loads and their timing. This prediction is robust: even for cases where the workload burstiness does not have a clear repetitive pattern, we still manage to achieve remarkable accuracy and outperform ARIMA and generic neural network models for future time windows that can range from 1 hour to a week. There are many important implications of this work:

Dynamic VM consolidation driven by resource demands of different percentiles: PRACTISE can provide resource usage predictions for VMs but also for physical machines (PMs) where the various VMs may be consolidated. PRACTISE can provide different types of usage statistics, e.g., means, and percentiles. Indeed, for PMs, the traces provide the number of VMs per PM, as well as resource usage information. PRACTISE can be used to predict this information and drive different consolidation strategies[5] that are based on different load statistics. Due to the remarkable accuracy of PRACTISE on capturing the peak loads and their timings, the consolidation policy can aggressively conserve resources without risking performance degradation.

VM consolidation driven by prediction of multiple resources: PRACTISE is able to explore the availability of both CPU and memory on PMs. Most importantly, having predictions on multiple resources, one can focus on the most scarce resource and drive the consolidation policies accordingly. Even more specifically, since there is significant burstiness in both memory and CPU usage, a strong prediction model can really help in optimizing usage for both resources.

Minimizing the impact of VM migration: the VM migration overhead is known to be non-negligible and application performance can thus drastically degrade, especially when the migration timing collides with peak loads of the VMs or the underlying physical hosts. Intelligent migration can greatly leverage the information of future loads provided by PRACTISE and select the optimal timings for migrating VMs.

A bird's eye view of data center resource usage: beyond VM consolidation, accurate information on future loads of different resources enables a holistic load management of the entire data center, including IT, cooling, and energy costs. The server loads consume the energy to power up not only server resources but also the cooling facility. Fundamental

[5]PRACTISE shows excellent prediction results for the PMs in this data set. These results are not shown due to lack of space.

questions, such as turning on/off components, can not be addressed without a holistic view of data center resource usage.

Other workloads: PRACTISE an be applied to any workloads with autocorrelation, e.g., storage workloads [121].

5.6 Chapter Summary

In this chapter, we develop PRACTISE, an enhanced neural network based framework for predicting the usage of various resources in data centers. PRACTISE uses autocorrelation-based feature selection, boostrap aggregation, and online updating. We extensively evaluate PRACTISE on predicting CPU, memory, disk and network usage on a set of 775 VMs over a period of 2 months and compare its prediction effectiveness to ARIMA and basic neural network models. We are able to achieve up to 3 times better prediction accuracy in terms of average prediction errors and dramatic improvements (2- to 9-fold) with respect to the prediction timings. Thanks to the excellent prediction accuracy of PRACTISE, we are able to efficiently capture the peak loads in terms of their intensities and timing, in contrast to classic time series models.

Chapter 6

Active Sizing for Resource Isolation in Data Centers

In the last chapter, we detailed introduce a neural networks based prediction method for data center resource usage series. This allows us to proactively allocate resources in the cloud data centers. Consequently, in this chapter, we propose an active sizing algorithm using the usage series prediction to resolve the performance tickets in cloud data centers. Performance ticketing systems provide the means to data centers to interactively improve user experience, maintain performance at tails, and guarantee smooth system operation. Typically, system monitoring and users issue tickets when encountering an array of performance violations, e.g., unresponsive service, high resource usage due to transient load dynamics, or persistent insufficient provisioning. Ticket resolution is unfortunately very expensive [68, 49] as a significant amount of manual labor is required for root-cause analysis and to remedy the detected problem [48]. Prior work has shown that there is strong correlation of ticket issuing with resource usage exceeding certain predefined thresholds [15]. In today's data centers, with physical resources being aggressively multiplexed across multiple virtual machines (VMs), the likelihood of issuing performance tickets due to physical or

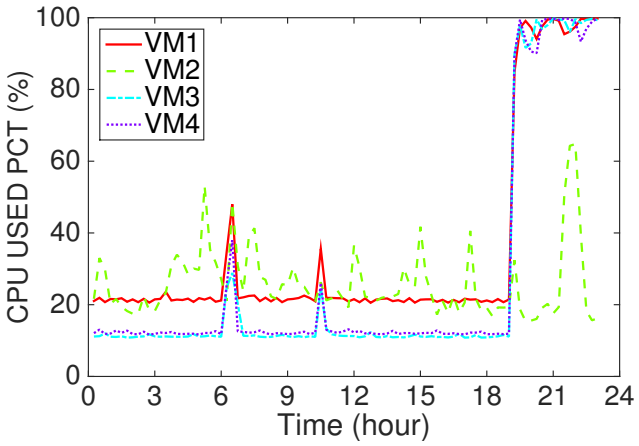


Figure 6.1: An illustration of spatial dependency across usage time series for 4 VMs co-located on a box.

virtual machines crossing predefined usage thresholds dramatically increases.

Past work has established that resource usage at data centers exhibits strong temporal patterns [16, 14]. Beyond temporal dependencies that are established by usage time series [120], it is common for co-located VMs to simultaneously compete for the limited physical resources, essentially exhibiting strong *spatial* dependency. We illustrate a motivating example in Figure 6.1 depicting the CPU usage time series[1] of 4 VMs co-located within the same physical box, where performance tickets are issued automatically when a VM utilization exceeds a threshold of 60%. One can easily see the spatial dependency of VMs 1, 3, and 4, i.e., time usages move up and down synchronously, and their respective tickets are triggered together, at around the 19:00 hour mark. These time series come from a data center production system and are quite representative of typical patterns in such systems. The temporal and spatial dependencies among VMs not only increase the number of tickets but also the difficulty in identifying their root cause and the corresponding resolution.

[1]We interchangeably use the terms time series and series.

The focus of this chapter is to develop a methodology to increase the data center dependability by using a *proactive* approach: reduce the number of tickets by predicting when they will occur in the future and by employing dynamic virtual machine resizing to adjust resource usage to avoid the triggering of future tickets. To this end, we first do a detailed, post-hoc workload characterization study of usage time series in production data centers of a major vendor which correspond to 80K VMs hosted on 6K physical servers. We develop an Active Ticket Managing (ATM) system that predicts future VM resource usage and proactively resizes the virtual resources of the resident VMs. The research challenges are numerous and outlined as follows.

Effective usage prediction is prerequisite to the development of any management policy. Indeed, in our past work we have shown that neural networks can be effectively employed for prediction [120], but their effective usage remains prohibitively expensive in practical situations as it suffers by its high training cost. In practice, in a large-scaled data center, with more than tens of thousands of physical boxes and hundreds of thousands of VMs, it is infeasible to rely on neural networks to predict future resource usage. We solve this first problem by developing a prediction methodology that discovers spatial dependencies across usage series and exploits them to develop an agile methodology for prediction. To this end, we introduce the concept of *signature VM series*, a subset of usage series that are representative of all other usage series. We are able to predict usage series not in the signatures set and the usage violation tickets of co-located VMs, via a linear combination of signature VM series, which provides a time series prediction model with as low as only 26% of the original time series. Second, based on predicted resource usage, we define a multi-choice knapsack problem and develop a greedy algorithm to dynamically adjust virtual resource allocation across co-located VMs. ATM is evaluated on production traces of 80K VMs and a small test-bed deployment on a cluster that runs MediaWiki [6], the

open source platform for Wikipedia. Our extensive evaluation results show that ATM has remarkably high accuracy in prediction, i.e., reaching prediction errors as low as 20% and significant ticket reductions, i.e., up to $60\% - 70\%$ less tickets while using only 26% of the original time series. The contributions of this chapter are as follows:

1) We do post-hoc characterization of usage ticket issuing in a large data center setting. We focus on discovering the distribution of usage tickets and spatial patterns of resources usages across co-located VMs. We find that usage tickets are mostly contributed by a small set of VMs, and that VMs show significant cross correlation among their CPU and RAM usage series.

2) Motivated by the strong spatial patterns across resources and co-located VMs, we argue that a small number of signature usage time series can be used as predictors to represent well the entire set of resource usage time series. This prediction methodology is the basis of ATM.

3) We develop a VM resizing policy to reduce usage tickets by setting the upper limits of CPU and RAM allocations when several VMs are co-located, a problem which is shown to be NP-hard. We rigorously formulate the ticket minimization problem subject to the physical capacity constraints. We propose a greedy algorithm to solve it and compare its performance to the max-min fairness algorithm.

The outline of this chapter is as follows. Section 6.1 discusses on the related work. Section 6.2 provides a characterization study on the usage tickets as well as the spatial patterns among usage series of co-located VMs. We propose spatial-temporal prediction methods for demand series in Section 6.3. In Section 6.4, we formulate the ticket minimization problem and demonstrate a greedy resizing algorithm to reduce usage tickets. An extensive evaluation of ATM on both production traces and a Wikipedia cluster is discussed in Section 6.5. Section 6.6 summarizes and concludes this chapter.

6.1 Related Work

Ticketing systems are widely used to improve on system dependability, e.g., slow responsiveness, failure [15], software bugs [73, 85] and system misconfigurations [126]. Prior art in ticketing systems centers on two directions: derive system management for software concurrency [73], database systems [48], and distributed data-intensive systems [125] but also to develop automatic detection systems for different types of tickets, bugs [85] and software misconfigurations by leveraging the rich correlation between configuration entries [126]. Machine learning has been used for automating ticket resolution recommendation [129, 98, 19]. To the best of our knowledge, there are no proactive methodologies for preventing ticket issuing, with the exception of models for database reconfiguration [50]. The proposed ATM policy fills this gap by not only deriving management insights for usage ticket patterns, but also by developing novel prediction and ticket avoidance strategies using VM resizing.

Time series prediction and analysis have been viewed as an excellent way to develop proactive system management policies [109, 132]. Temporal models such as ARIMA models [23] have been widely used to predict time series with strong seasonality. Sophisticated neural network models show a strong promise in capturing highly irregular time series at a cost of long training overheads [72]. Time series clustering aims to explore spatial dependency, either through their original series, e.g., DTW [13], or extracted features [43], e.g., moments. ATM combines spatial with temporal models to contain the cost of neural network training and scales well for very large numbers of time series.

Virtualization technology has become the industry standard offering great opportunities to multiplex physical resources over a large number of VMs. There are two ways to change the efficiency of resource multiplex ratios: by sizing the virtual resource capacities [101] and by dynamically consolidating VMs [115]. While dynamically changing the

degree of VM consolidation is shown effective to take advantage of the time variability of the workload [36], the overhead of migrating VMs can greatly reduce its performance benefits. On the contrary, sizing resource of co-located VMs incurs less system overhead [101]. A central question of multiplexing resources is how to strike a good tradeoff of fairness and performance for workloads, e.g., latency [54] and throughput [114]. Fairness driven policies, e.g., max-min fairness, proportional fairness, and bottleneck resource fairness [18], have been proposed for various systems components, including storage systems [114] and networks [102]. The sizing algorithm proposed in ATM differs from related work by its objective to reduce the number of usage tickets. While max-min fairness also reduces the number of tickets, it cannot achieve this as effectively as ATM since ticket reduction is a side-effect rather than a main focus.

6.2 Statistics and Observations

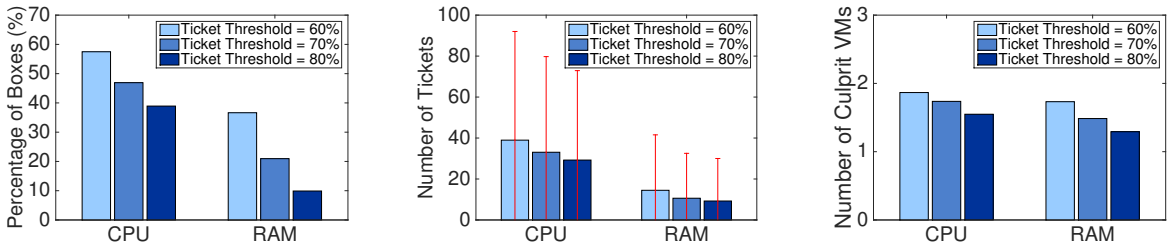
The motivation for the design of ATM is the urge to reduce usage tickets that are typically issued when VM resource utilizations exceed certain thresholds. The trace that we consider here comes from IBM production data centers serving various industries, including banking, pharmaceutical, IT, consulting, and retail, and using various UNIX-like operating systems, e.g., AIX, HP-UX, Linux, and Solaris. The majority of VMs in the trace are VMware VMs. The trace contains CPU and RAM capacity but also utilization data taken at a time granularity of 15 minutes for 6K physical boxes hosting more than 80K VMs during a 7-day period from April 3, 2015 to April 9, 2015. Naturally, the level of consolidation is very high, i.e., on average 10 VMs are consolidated within a single physical box [16]. In addition, both VMs and boxes are very heterogeneous in terms of resource configuration.

In the following, we first show the distribution of usage tickets under different ticket thresholds, followed by a more detailed analysis on the spatial patterns of usage series of

co-located VMs. We aim to uncover how usage tickets are distributed across resources and most importantly how usage patterns trigger usage tickets. We anticipate that the design principles of the proposed ATM system leverages this characterization analysis.

6.2.1 Usage Tickets

Usage tickets are generated when utilization values exceed target thresholds. Naturally, lower thresholds trigger a higher number of usage tickets and increase the cost of resolution, whereas higher thresholds result into fewer tickets but at a higher risk of performance degradation. To quantify the effect of different thresholds, we consider three threshold levels, namely 60%, 70%, and 80%. Such values are commonly adopted in production systems [19]. Figure 6.2 illustrates quantitative information on the issued tickets for the CPU and RAM usage series of April 3, 2015. We focus on the following: how many boxes have tickets and how these tickets are distributed across co-located VMs and their resources.



(a) PCT of boxes w/ tickets (b) Distr. of tickets (c) Number of culprit VMs

Figure 6.2: Characterization of usage tickets for CPU and RAM of VMs per box.

Figure 6.2(a) plots the percentage of boxes that have at least one VM usage ticket under the different thresholds. Even with the highest ticket threshold of 80%, almost 40% of boxes obtain at least one ticket due to CPU violation and 10% due to RAM violation, these percentages increase to 57% and 38%, respectively, when the threshold is

60%. Overall, the percentage of boxes having CPU tickets is higher than RAM tickets, independently of the threshold. This can be explained by the fact that RAM tends to be over-provisioned for performance reasons. Figure 6.2(b) illustrates the mean and standard deviation of the number of tickets per box for CPU and RAM. The average number of CPU(RAM) usage tickets per box are 39(15), 33(11), 29(9), for the three thresholds of 60%, 70%, and 80%, respectively, showing a relatively minor decreasing trend. The next natural question is whether tickets are evenly distributed across all co-located VMs. To this end, we compute the number of VMs that accounts for the majority of tickets, where the majority is defined to 80% of usage tickets per box (this is an ad-hoc value). Figure 6.2(c) shows that on average one to two VMs per box cause the majority of tickets irrespective of the three threshold values. A further interesting observation is that since the culprit VMs are few, if we increase the capacity allocation of the culprit VMs by removing resources from other co-located VMs, then tickets may reduce. On the contrary, if tickets are evenly distributed, resizing does not help.

6.2.2 Do Spatial Dependencies Exist?

To better understand the spatial patterns of usage tickets, we estimate the magnitude of spatial dependency by computing the Pearson's correlation coefficients [23] over each pair of CPU and RAM usage series of co-located VMs. For each box and co-located VMs, we compute four types of correlation coefficients ρ: (i) between any pair of CPU usage series (intra-CPU), (ii) between any pair of RAM usage series (intra-RAM), (iii) between any pair of CPU and RAM usage series (inter-all), and (iv) between CPU and RAM usage series from the same VM (inter-pair). The first two correlation metrics measure the relationship among specific resources, i.e. CPU and RAM, time series ("intra" resource measures), the latter two focus on the relationship between CPU-RAM pairs ("inter" measures). For

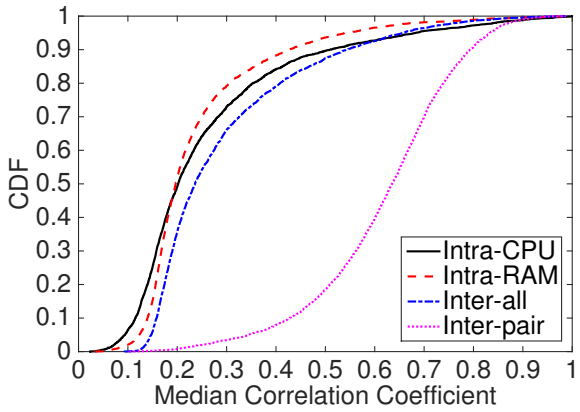


Figure 6.3: Cumulative distribution of correlation of intra-CPU, intra-RAM, inter-CPU/RAM.

each box, we compute the median value of the above measures and present the cumulative distribution functions (CDFs) across all the boxes in Figure 6.3.

One can immediately see from the shapes of the CDFs that intra-RAM ρ is higher than intra-CPU, followed by inter-resources measured from any pair of VM or the same VM. This implies that inter-resource dependency is higher than the intra-resource one. Indeed, the mean values for intra-CPU, intra-RAM, inter-CPU/RAM from any pair, inter-CPU/RAM from the same VM are 0.26, 0.24, 0.30, and 0.62 respectively. The CDFs give a clear message: the CPU-RAM pairs across co-located VMs are correlated, this is a fact that we take advantage of when we attempt to use clustering to reduce the cost of prediction.

6.3 Spatial-Temporal Prediction Models

We first elaborate on the challenges for concurrent prediction for a large number of time series representing multiple resource usages from co-located VMs at production data centers. The immediate obstacles of prediction given a large number of demand series are accuracy, training overhead, and model scalability. Typically, temporal models [23], such as ARIMA are not able to capture well bursty behaviors. More sophisticated temporal models such

as neural networks, capture irregular patterns better but at much higher computational overheads. Given such restrictions, it is important to come up with efficient and accurate prediction models that also scale well.

We propose a new prediction methodology that combines both temporal and spatial models to predict on each box the resource demand time series[2] $D_i\,(\forall i \in [1, M \times N])$ where M is the number of co-located VMs and N is the number of different resources taken into consideration. We introduce the concept of *signature series:* a minimum number of time series that are predicted via *temporal models*. The rest of the demand series, termed as *dependent series*, are predicted through a linear combination of signature series via *spatial models*. Essentially, we divide the demand series, D_i, into two sets: the signature set, denoted by Ω_s, and the dependent set, Ω_d.

The novelty of ATM is to derive novel spatial models for dependent series while applying existing temporal models to predict signature series. Many practical techniques exist in the literature for reducing the overhead of temporal models by extracting and storing features of the time series [120, 43]. We stress that any temporal prediction model can be directly plugged into the ATM framework.

To derive the spatial models, we want to express all demand series $D_k, k \in \Omega_d$ by a linear combination f_k of the signature series $D_j, j \in \Omega_s$:

$$D_k = f_k(D_j). \tag{6.1}$$

As every demand series can be either a signature or a dependent series, a brute force solution to find the minimum signature set is to explore all $2^{N \times M}$ combinations of regression models. For boxes hosting an average number of VMs, i.e., M around 10 and expected

[2]Demand series is the product of usage series and the allocated virtual capacity. Both demand and usage series share the same correlation characteristics. For the purpose of virtual resource resizing, we predict demand series directly.

to grow as servers become more powerful, it is clear that this method is not viable. To address this issue, we devise an efficient searching algorithm that can quickly find signature series without using exhaustive search, by leveraging time series clustering techniques and stepwise regression.

6.3.1 Searching for Signature Demand Series

Key to the discovery of signature series is clustering. We propose a two-step algorithm to identify the signature set Ω_s. *Step 1* defines the initial set of signature series. This is achieved using time series clustering, specifically dynamic time warping (DTW) [13] or correlation based clustering (CBC) that we propose here. *Step 2* defines the final set of signature series by detecting and removing multicollinearity among initial set of signature series using variance inflation factors (VIF) and stepwise regression. The intent of the second step is to fix the pitfall that although signature series appear independent, it is possible that a combination of certain subsets of the initial signature series can well represent the others. For example, a group of series can be separated into three clusters because of their dissimilarity in the distances or the correlation patterns. If however one of the clusters can actually be well expressed as a linear combination of the other two, then this falls under a classical example of multicollinearity. Figure 6.4 illustrates the steps of signature set search.

Step 1: Time Series Clustering

Dynamic time warping is an effective solution for finding clusters of time series where the distance across the series is short. A potential problem is that DTW falls short in capturing within the cluster series that are of larger distance. Correlation based clustering solves this problem by capturing highly correlated time series that are far enough apart and cannot

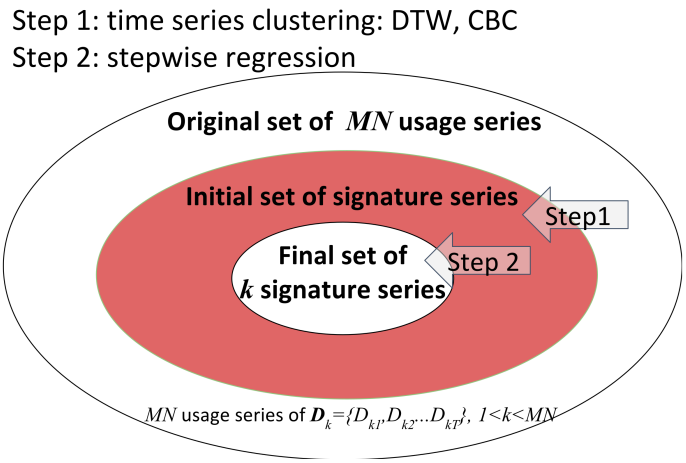


Figure 6.4: Overview of searching for signature set.

be captured by DTW. Applying DTW on the exemplary four series shown in Figure 6.1 illustrates how clustering with DTW only offers a partial solution. DTW detects three clusters: cluster 1 VM1, cluster 2 VM2, and cluster 3 VM3 and VM4. CBC instead puts VM1, VM3, and VM4 within the same cluster. Indeed, the series D_1 and D_4 of VM1 and VM4 can be well represented as linear models of the series D_3 of VM3, e.g., $D_1 = a_0 + aD_3$, and $D_4 = b_0 + bD_3$, where a_0, a, b_0, and b are scalars. In the remaining of this section we provide details on DTW and CBC.

Dynamic Time Warping Clustering: The high level idea of DTW is to group series that show low *distance dissimilarity*. To obtain the distance dissimilarity between two series $P = \{p_1, p_2, ..., p_i, ..., p_n\}$ and $Q = \{q_1, q_2, ..., q_j, ..., q_m\}$, we first build a matrix that consists of the pair-wise squared distances, i.e., $d(p_i, q_j) = (p_i - q_j)^2$, between each pair of elements p_i and q_j in the two series. The distance dissimilarity $\lambda(n,m)$ of the two series is given by the wrapping path through the matrix that minimizes the total cumulative distance [13] and can be recursively computed as follows:

$$\lambda(i,j) = d(p_i, q_j) + min\{\lambda(i-1, j-1), \lambda(i-1, j), \lambda(i, j-1)\}. \quad (6.2)$$

Next, we apply hierarchical clustering [91] for any given number of clusters, ranging from 2 to $(M \times N)/2$ since we aim to reduce the original set to at least its half. We determine the optimal number of clusters, based on the average silhouette value [94] of all time series within each cluster. For each series i, its silhouette value $s(i)$ is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{b(i), a(i)\}} \tag{6.3}$$

where $a(i)$ is the average distance dissimilarity between series i to all the other series within the same cluster using DTW, and $b(i)$ is the lowest average distance dissimilarity between series i to the all the series in a different cluster. The higher the silhouette value, the better the series lies within its cluster. For each number of clusters, we average the silhouette values of all the series as the representative silhouette value. The optimal number of clusters is the one with the maximal silhouette value. As last step and beyond conventional DTW, we identify each signature series as the series with the lowest average dissimilarity in each cluster.

Correlation-based Clustering: CBC focuses on grouping series showing high correlation. For each box, we first compute the pairwise correlation coefficients, denoted as ρ, for all pairs of the $M \times N$ series. For a demand series D_i, there are $(M \times N - 1)$ pairs $\rho_{i,l}, \forall l \neq i$. To form the clusters, we rank each series $D_i, i \in [1, M \times N]$ first by the total number of $\rho_{i,l}$ above a threshold ρ_{Th}, and second by the mean value of the $\rho_{i,l}$ above the threshold. In the following we set $\rho_{Th} = 0.7$, a common threshold value used to determine strong correlation between two series, which suggests a potential for linear fitting [60]. After the series have been ranked, we select the topmost one and remove it together with all the series being correlated with it with a correlation coefficient higher than the threshold. These series are now considered within a new cluster with the top ranked series being the signature series. This procedure continues by selecting the next topmost series still in the ranked list and

ends when the ranked list becomes empty.

Step 2: Stepwise Regression

To further reduce the number of signature series, we calculate the variance inflation factor – a metric that can detect multicollinearity in regression. For each series in the signature set, we regress it on the rest of signature series and obtain its VIF value [60]. The rule of practice is that a VIF greater than 4 indicates a dependency with the other series in the initial set. After detecting the risk of multicollinearity, i.e., at least one series has a VIF greater than 4, we perform standard stepwise regression to remove the series that can be represented as linear combinations of the other signature series.

6.3.2 Prediction Models

To predict all $M \times N$ demand series, we first predict the signature series $D_i\,(i \in \Omega_s)$, using neural network models and their historical data [120]. To predict all dependent series, we regress each dependent series on the set of signature series, obtaining coefficients using ordinary least square estimates. We stress that the signature series predictions are not tied to the any specific model rather any suitable prediction model can be easily plugged into our ATM framework.

In summary, we first leverage historical data to develop spatial models to define dependent series and their respective signatures. Later, we use temporal models to predict the signature series and inexpensive linear transformation models to predict the dependent series.

6.3.3 Results on Spatial Models

Prior to moving to the proposed VM resizing policy, we present evaluation results of the proposed spatial models across the demand series of the trace data (6K boxes and 80K VMs) presented in Section 6.2. Our evaluation focuses on: (i) the difference between DTW and CBC clustering, (ii) the effectiveness of clustering and stepwise regression, and (iii) inter- v.s. intra-resource models, i.e., if it is necessary to treat different resource series, e.g., CPU and RAM, separately. Since the purpose of spatial models is to use a minimum subset of original series to accurately represent the data center, the metrics of interest are: (i) the percent of signature series out of the total demand series and (ii) the prediction error. In this section we only focus on the effectiveness of the spatial models, i.e., how close the dependent series are from the actual time series counterparts. The overall prediction accuracy of combining spatial models with temporal models is presented in Section 6.5.

6.3.3.1 Difference between DTW and CBC

Figure 6.5 compares the distribution of the number of clusters resulting from DTW and CBC and highlights the type of each signature series, i.e., CPU or RAM. For DTW, roughly 70% of boxes have only 2 to 3 clusters, and the rest have 4 to 31 clusters. In contrast, CBC is less aggressive resulting in a higher number of clusters and, consequently, a higher number of signature series. This indicates a higher overhead to develop their temporal models. Moreover, in terms of signature series types, under DTW, one can see that both CPU and RAM roughly account for 50% of the signature series. This is consistent across all DTW bars in Figure 6.5. Instead, with CBC, most signature series are CPU series.

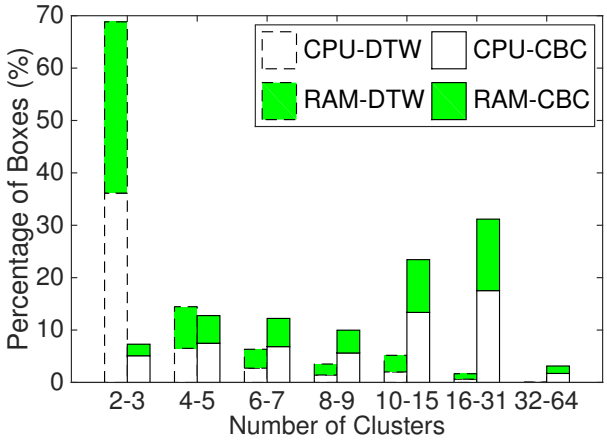


Figure 6.5: Comparison of clustering results using DTW and CBC.

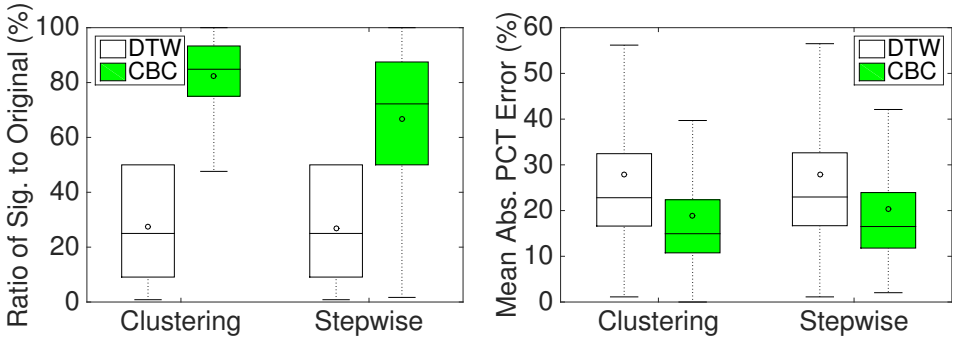
6.3.3.2 Effectiveness of the Two-Step Approach

To better illustrate the benefits of time series clustering (DTW or CBC) and stepwise regression, we compare the signature set reduction and prediction accuracy of each step in Figure 6.6. Each box represents the 25^{th}, 50^{th} (mid line), and 75^{th} percentiles, whereas the dot marks the mean and the whiskers the most extreme data points.

Figure 6.6(a) shows the percent of signature series out of the total number of series for each of the 6K physical boxes. Since DTW is quite aggressive in reducing the number of time series, there is almost no further reduction after applying stepwise regression. Both steps reduce the entire set to 26%. After CBC, the set is reduced to 82%, however stepwise regression brings further down the number to 66%.

Considering prediction accuracy, as shown in Figure 6.6(b), both DTW and CBC experience minor losses. The average absolute percentage error (APE)[3] from DTW is about 28%, while the average APE for CBC is only around 20%. Since stepwise regression almost does not affect the signature set of DTW, one expects no obvious decrease in prediction

[3]APE is defined as $APE = \frac{|Actual - Fitting|}{Actual}$



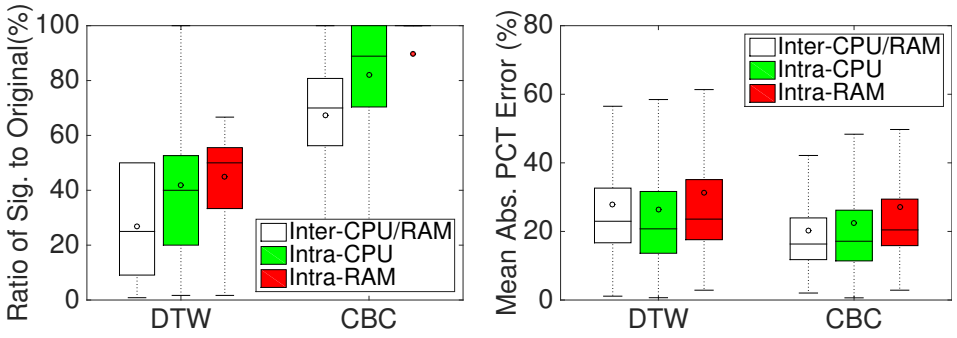
(a) Signature set reduction (b) Prediction error

Figure 6.6: Comparison of the two steps: effectiveness of clustering and stepwise regression.

accuracy. This is indeed shown in the graph. Surprisingly, the same holds true with CBC where stepwise regression reduces CBC's accuracy only by 1%. These results confirm the effectiveness of stepwise regression in reducing the signature set without degrading in prediction accuracy.

6.3.3.3 Inter- v.s. Intra-Resource Models

We compare the effectiveness of the proposed inter-resource models, i.e., combining CPU and RAM as predictors, against the intra-resource models, in which CPU and RAM are treated separately. In Figure 6.7, we summarize the prediction errors and reduction in the original demand series. Inter-resource models can not only reach a lower prediction error but also a higher reduction in the number of demand series, than intra-CPU and intra-RAM models. We present the results in box plots. In terms of average APEs of prediction for CBC(DTW), the inter model is around 20%(28%), whereas the intra-CPU and intra-RAM are 21%(26%) and 23%(31%). Again, these results are in good agreement with our observation in Section 6.2 that inter resource correlation is higher than intra-CPU and intra-RAM correlations. In terms of the average number series in the signature set for CBC(DTW), the inter model uses roughly 66%(26%) of the original series, while



(a) Signature set reduction (b) Prediction error

Figure 6.7: Comparison of inter- and intra-resource models.

intra-CPU and intra-RAM can use up to 81%(41%), 90%(45%) compared with the original set. Overall, the inter model can greatly benefit from the correlation across co-located resources.

6.4 Virtual Resource Resizing

Being able to accurately predict future usage enables the very first step to actively manage usage-related tickets. Having future usage knowledge, it is possible to develop a virtual resource resizing policy that can effectively reduce the number of usage tickets. The monitoring systems in modern data centers track the resource usages at discrete windows, e.g., 15 minutes, termed as the ticketing window, and compare them with ticket thresholds to determine whether a ticket needs to be issued or not. To avoid incurring overreaction to transient loads, we set the resizing window to be greater than the ticketing window. For the data centers considered here, ticket resolution occurs within a day of the ticket being issued, so setting the resizing window to one day is a reasonable assumption. This implies that the prediction horizon of the demand series needs to be also one day. Note that past work has shown that the accuracy of prediction decreases as the prediction horizon

increases [120], so setting the prediction window to such a high value makes ATM more conservative than it can actually be. During each resizing window, ATM devises and actuates the virtual resource allocation of co-located VMs on boxes. The objective is to find optimal sizes for co-located VMs to achieve the lowest number of tickets, subject to various resource constraints at boxes. The resources considered are: virtual CPU measured in GHz and virtual RAM measured in GB.

There exist a large body of virtual resource allocation studies aiming to satisfy various performance targets, e.g., user response time, system utilization, and fairness. Max-min fairness [106, 46] is one of the most applied allocation policies that tries to guarantee the performance of small VMs, given the assumption of known demands. Our resizing problem can be viewed similarly but with the objective to minimize the occurrences of target utilization threshold violations. We develop a resizing algorithm based on a rigorous optimization formulation, which is later transformed into a multi-choice knapsack problem (MCKP) with tunable discretization parameters. The introduction of such discretization parameters enables us to reduce the complexity and increase the safety margin in resource allocation. In contrast to spatial-temporal prediction models, the resizing algorithm treats CPU and RAM separately due to separate constraints on each resource. Hence for simplicity, in the following we redefine the index i in D_i to be the index of a VM rather than the index of a specific resource on a VM.

6.4.1 Ticket Optimization Formulation

We formally introduce the problem, including notations and constraints, for resizing all co-located VMs on a single box. The foremost important constraint is that the summation of allocated virtual resources should be less than or equal to the total available virtual resource, i.e., $\sum_i C_i \leq C$, where C_i denotes the virtual capacity allocated to VM i, and C is

the total available virtual capacity at the box. The decision variable is C_i and needs to be determined at the beginning of the resizing horizon.

The prediction module provides all demand series values for the entire resizing window, equal to T ticketing windows, for VM i, $D_i = \{D_{i,1}, \ldots D_{i,T}\}$. We introduce an indicator variable, $I_{i,t}$, when $I_{i,t} = 1$ a usage ticket occurs to VM i at ticketing window t, because the demand exceeds a certain threshold of the capacity, say, αC_i (e.g., $\alpha = 0.6$); otherwise $I_{i,t} = 0$. We aim to minimize the total number of tickets occurring on all co-located VMs during the resizing window. Thus, we can write the objective function as $\sum_i \sum_t I_{i,t}$. In summary, we can define the ticketing optimization problem as:

$$(\mathbb{R})\ \min \quad \sum_i \sum_t I_{i,t} \tag{6.4}$$

$$s.t. \quad \sum_i C_i \leq C \tag{6.5}$$

$$D_{i,t} - \alpha C_i \leq D_{i,t} I_{i,j} \tag{6.6}$$

$$I_{i,t} \in \{0, 1\} \tag{6.7}$$

Constraint (6.6) ensures that $I_{i,t} = 1$, when the demand exceeds the ticket threshold, αC_i; otherwise the objective function drives $I_{i,t}$ to zero. The problem $\mathbb{R}$ is a classical mixed integer linear programming (MILP), whose complexity greatly depends on the number of integer variables, i.e., the indicator variables $I_{i,t}$ in our case. The number of indicator variables for each box is thus the product of the number of ticketing windows, T, and the number of VMs, M.

6.4.1.1 Resizing Algorithm

Instead of resorting to a standard MILP solvers, such as CPLEX [3], we transform the original problem into a multi-choice knapsack problem by Lemma 6.4.1: the optimal size

for each VM must be equal to one of the demand values in D_i or 0. The advantages of transforming the original problem into a MCKP are twofold: (i) there exist a large number of efficient algorithms for MCKP and (ii) it allows for a reduction of the number of integer variables. We elaborate on the second point after formally introducing the transformation of the original optimization problem to MCKP.

Lemma 6.4.1 *For VM i, the optimal size $C_i* \in D_i \cup \{0\}, D_i = \{D_{i,1}, D_{i,2} \dots D_{i,T}\}$.*

Proof: If there exists an optimal solution (C_i*) for each VM (i) for the resizing problem, C_i* has to be in one of the three ranges: $[0, min\{D_i\})$, $[min\{D_i\}, max\{D_i\})$, and $[max\{D_i\}, +\infty)$. If C_i* is less than $min\{D_i\}$, we argue that C_i* could be set to 0 and the objective function stays unchanged while the constraints are not violated. Similarly, it is proven that if C_i* is not less than $max\{D_i\}$, C_i* can be set to $max\{D_i\}$. If C_i* is in $[min\{D_i\}, max\{D_i\})$, sort D_i in a descending order as $D_i^{descend} = \{O_1, O_2, ..., O_p, O_{p+1}, ...\}$. Following the same reasoning, it is possible to determine that $\exists\, q, C_i* \in [O_q, O_{q+1})$. In addition, setting C_i* equal to O_q, the minimum objective function can be obtained without any constraint violation. Hence the optimal size C_i* is either in D_i or 0. ■

Based on Lemma 6.4.1, we can transform the original formulation into a multi-choice knapsack problem, whose complexity can be further simplified by reducing the number of indicator variables. We first introduce a reduced demand set with 0 added, denoted as D'_i, containing the unique values of the original demands in decreasing order, $D'_{i,v+1} \leq D'_{i,v}$. According to Lemma 6.4.1, one of them is the optimal capacity. We note that $D'_{i,v}$ is not the same as $D_{i,t}$. The following small example illustrates the difference. Given a specific demand series $D_i = \{30, 30, 40, 40, 23, 25, 60, 60, 60, 60\}$, its reduced series is $D'_i = \{60, 40, 30, 25, 23, 0\}$ containing only the unique values plus 0 in descending order.

We introduce a new binary variable $Y_{i,v}$, denoting that the unique value $D'_{i,v}$ is chosen to be the capacity for VM i. The next step to reduce the problem into MCKP is to define the

number of tickets, denoted $P_{i,v}$, seen by VM i when the value of $D'_{i,v}$ is chosen as capacity, i.e., $Y_{i,v} = 1$. Following the previous example of reduced demand set, we show an example of ticket calculation. Let us assume the current capacity is 70 and the ticketing threshold for issuing usage tickets is 60%. We thus know that demands greater than $70 \times 60\% = 42$ at any ticketing window will result into tickets. We can then obtain $P_i = \{0,4,6,8,9,10\}$. Due to the decreasing order of D'_i, P_i has an increasing order, i.e., $P_{i,v+1} \geq P_{i,v}$. The total number of tickets for a box can thus be written as $\sum_i \sum_v Y_{i,v} P_{i,v}$ and the resource constraint of the total capacity as $\sum_i \sum_v Y_{i,v} D'_{i,v} \leq C$.

In summary, we reach a multi-choice knapsack problem, where items (in the original knapsack problem) are divided into subgroups and exactly one item needs to be selected from each group. Putting our problem into the context of multi-choice problem, we have M groups of VM demands and we need to choose exactly one demand from each group as their capacity. The decision variables are $Y_{i,v}$ denoting that a particular demand is chosen as the size for VM i, where $i \in [1,M]$ and that the number of tickets, $P_{i,v}$, can be seen as "weights". The transformed ticket reduction problem is:

$$(\mathbb{R}')\ \min \quad \sum_i \sum_v Y_{i,v} P_{i,v} \tag{6.8}$$

$$s.t. \quad \sum_i \sum_v Y_{i,v} D'_{i,v} \leq C \tag{6.9}$$

$$\sum_v Y_{i,v} = 1 \tag{6.10}$$

$$Y_{i,v} \in \{0,1\} \tag{6.11}$$

The formulation of problem $\mathbb{R}'$ enables the introduction of a tunable parameter, ε, which decides the discretization of demand values. We illustrate this point using the running example of original series D_i and its reduced series D'_i. The original formulation $\mathbb{R}$ has 11 integer variables (including the 0), whereas the transformed problem $\mathbb{R}'$ has only

6 integer variables. One can even further decrease the number of binary variables in P_i by discretizing the demand values, such as rounding off the first digit. For example using $D'_i = \{60, 40, 30, 0\}$, where 23 and 25 are rounded up to 30. Another point worth mentioning is that we need to update the number of corresponding tickets too, i.e., $P_i = \{0, 4, 6, 10\}$. Rounding up demands makes the resizing algorithm more aggressive in allocating resources. Consequently, we formally introduce a discretization factor, ε, which further reduces the complexity and provides a safety margin for resource allocation. We note that ε is only applied on the predicted series. In summary, the initial step computes D'_i from D_i using ε, and calculates their corresponding tickets, P_i for all co-located VMs i.

To solve the MCKP problem, we resort to the so-called minimal algorithm [90]. We illustrate the general idea in the context of our resizing problem. The algorithm chooses capacity candidates for each VM and shuffles around the capacity across VMs, comparing to the available capacity and marginal ticket reductions. For all VMs, it chooses capacity candidates that can incur a minimum number of tickets, i.e., starts from the maximum values in D'_i. When there is no sufficient capacity to achieve such allocations for all VMs, the priority is given to the VM having the lowest marginal ticket reduction values (MTRV). MTRV represents the additional ticket increment when reducing one unit of capacity provisioning. Its formal definition is:

$$MTRV = \frac{P_{i,o} - P_{i,o-1}}{D'_{i,o-1} - D'_{i,o}}, \tag{6.12}$$

where o denotes the index of candidates in D'_i. The VM with the lowest MTRV is always chosen to reduce the capacity provision from its current candidate value to the next one in D'_i. Note that as D'_i is in decreasing order, the next candidate immediately implies a capacity reduction. Once the candidate list is updated, the same process continues until the sum of all candidates is less or equal to the available capacity.

For a practical implementation, in addition to the constraint of total available capacity, it is also imperative to consider the lower and upper bounds of capacity. In order to avoid spillovers of unfinished demands from previous ticketing windows, we impose a lower bound on the VM capacity size, such that its peak usage before resizing is satisfied. Moreover, as any VM is not able to use more resources than the available resource amount of the underlying physical box, we introduce the allocation upper bound based on the box resource capacity. We can easily incorporate such lower and upper bounds into our resizing algorithm by limiting the values in D'_i for each VM i.

6.4.2 Results on Usage Ticket Reduction

Prior to moving on to the evaluation of the full-fledged ATM, i.e., the combination of spatial-temporal prediction and resizing policy, we first show how effective the proposed resizing algorithm is against existing resource allocation heuristics. For a fair comparison, the demand inputs are based on the original dataset described in Section 6.2, instead of prediction. We implement the max-min fairness algorithm [106] and a "stingy" algorithm which only allocates the capacity according to the lower bound, i.e., the maximum demand regardless of the ticket threshold, often used in practice. In contrast, the max-min algorithm starts to allocate to all VMs the demand of the smallest VM, considering its ticket threshold, and continues onto VMs in the increasing order of their demands until all capacity is exhausted.

Here, we evaluate the data of April 3, 2015 across all 6K boxes and set the threshold to trigger usage tickets to 60%: i.e., every 15-minute ticketing window the monitoring system checks if the average usage of CPU or RAM of each VM exceeds the 60% of the allocated capacity. Figure 6.8 summarizes the mean ticket reduction (in percent) and its standard deviation, when applying the proposed ATM resizing, max-min fairness, and

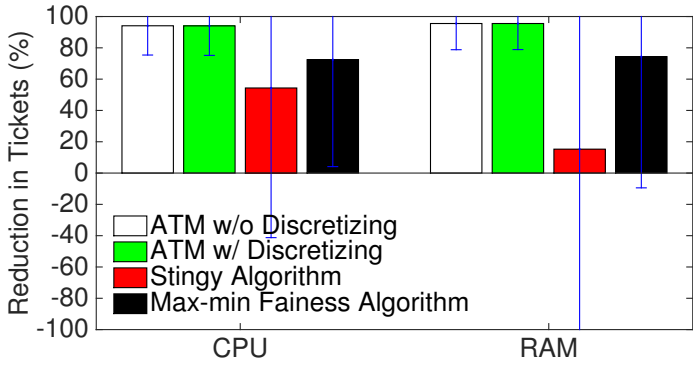


Figure 6.8: Ticket reduction for CPU and RAM: comparing ATM, max-min fairness, and stingy algorithms.

stingy algorithms. As expected, the stingy algorithm is completely unaware of the ticket threshold. On average it achieves a ticket reduction of 54% and 15% for CPU and RAM, respectively. Max-min fairness reduces the tickets by around 70% for both CPU and RAM. This is still roughly 30% worse than the ATM resizing results. Due to the nature of favoring small VMs, large VMs can be severely punished under max-min fairness resulting in no ticket reduction and explains the high standard deviation under max-min fairness.

As a pleasant surprise, our resizing algorithm does exceptionally well. It achieves 95% and 96% usage ticket reductions for CPU and RAM, respectively, a remarkable improvement for both performance and cost. This is also attributed to the fact that the systems of the original traces are equipped with abundant resources, i.e., typically data centers are lowly utilized [16]. By simply shuffling resources across co-located VMs, we are able to achieve significant performance gain. Moreover, we also eliminate the overhead of inspecting and resolving a large number of usage tickets, a process that is known to be expensive.

6.4.3 Actuation of Virtual Capacity

Cloud data center tenants are typically charged by the amount of virtual resources, for example, the number of virtual cores. Consequently, any practical sizing policy should

adhere to such a constraint, due to accounting and financial concerns. Therefore, to enforce the virtual capacity limits decided in our algorithm, we use the control groups (cgroups) feature of the Linux kernel [2]. Cgroups allow to limit, account for, and isolate resources usages of groups of processes. By placing the processes and threads relating to each VM in a separate cgroup, we can dynamically change the resource usage limits for each VM. To simplify the cgroups configuration, we expose the resource limits through a web-based API by running a small daemon at each hypervisor. The advantage of cgroups over directly modifying the allocated virtual VM resources is that the latter typically requires a restart of the guest OS while the former can be changed on-the-fly without disrupting the VM operation. Moreover, cgroups offer a finer-grained CPU control with an almost continuous CPU limit control rather than the stepwise decrease/increase of virtual cores.

6.5 Evaluation

We extensively evaluate ATM not only on a large number of data center production traces but also experimentally on a cluster running MediaWiki. We focus on presenting the effectiveness of ATM in ticket reduction to improve system dependability and to reduce the high cost associated with ticket resolution. In the remaining of this section, we assume that usage tickets related to CPU and RAM are automatically issued when VM utilization is greater than 60%.

6.5.1 Production Systems

We focus on a subset of boxes from the data center trace (400 boxes) which have no gaps in their traces. The remaining box traces suffer, throughout the 7 days of the trace, from occasional gaps with no data. We show how different configurations of ATM can proactively reduce the number of tickets. We engage training of the signature series for

5 days and then apply ATM and VM resizing for the following day. We stress that this analysis is post-hoc, i.e., we can not change the size of the actual VMs in the trace, we focus only on the prediction accuracy and ticket reduction via ATM. On the contrary, in the experimental evaluation on the MediaWiki cluster presented in the Section 6.5.2, we do also illustrate VM resizing in a working system.

For the spatial models, we consider DTW and CBC clustering techniques and set the discretization factor $\varepsilon = 5$. The temporal models used for the signature series are neural networks [120]. ATM performs the prediction of 16000 usage series, each of which has 96 ticketing windows, with each window being 15 minutes long. After obtaining the predicted series, ATM triggers the resizing algorithm for every box to determine the near optimal CPU and RAM capacity for all co-located VMs. We note that results presented in this section differ from Section 6.3 and 6.4, where only the proposed spatial models and resizing algorithms are evaluated individually, excluding the temporal prediction models. Here, we have the full effect of both prediction models.

6.5.1.1 Prediction Errors

Figure 6.9 presents the CDF of the prediction accuracy of ATM in terms of APE with different spatial models, i.e., DTW and CBC clustering. For CPU and RAM usage, we use the inter-resource model, i.e., signature series are a mix of CPU and RAM. The average prediction errors of resources usage per box are 31% and 23%, for DTW and CBC, respectively. These are only slightly higher than the errors without the temporal models presented in Section 6.3. The figure also illustrates the CDF of the mean absolute errors for peak demands, i.e., usage higher than 60%. The average peak errors across all boxes are 20% and 17% for DTW and CBC, respectively. This shows that neural networks can capture well the temporal dynamics of the signature series. We further note that this high

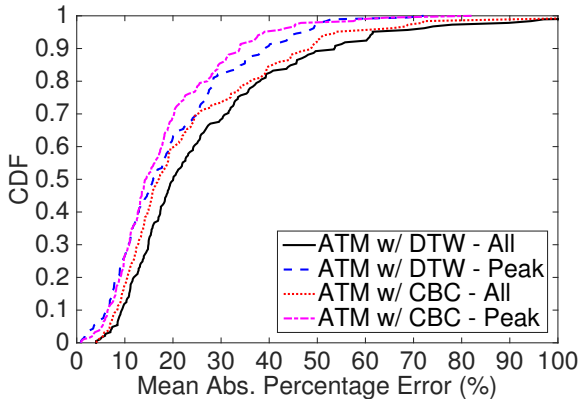


Figure 6.9: CDF of prediction accuracy of ATM on 400 production servers: ATM_{CBC}, ACM_{DTW}.

accuracy of temporal models is achieved at a high computational time and long historical data, i.e., 5 days, whereas the prediction of dependent series via spatial models has a negligible cost. We also note that the reduction in demand series for this subset of 400 boxes is similar to results shown in Section 6.3 across 6K boxes.

6.5.1.2 Ticket Reduction

Figure 6.10 compares the results of average ticket reduction using two different versions of ATM against the max-min fairness, and stingy policies, see Section 6.4. Each bars illustrate the mean and standard deviation of ticket reduction across boxes divided into CPU and RAM tickets. The key observations are the following. Both versions of ATM are able to achieve a higher ticket reduction, around 60% and 70% for CPU and RAM, respectively, compared to the other two heuristics. We like to point out that the standard deviation is high for all four strategies indicating huge difference across boxes. Different from the resizing results shown in Section 6.4, max-min fairness shows worse reduction results than stingy. This can be explained by the observed high variability across the chosen 400 boxes which shows that max-min fairness could even result in a increase of the number of tickets for a subset of the boxes, see the range of standard deviation. Max-min fairness favors

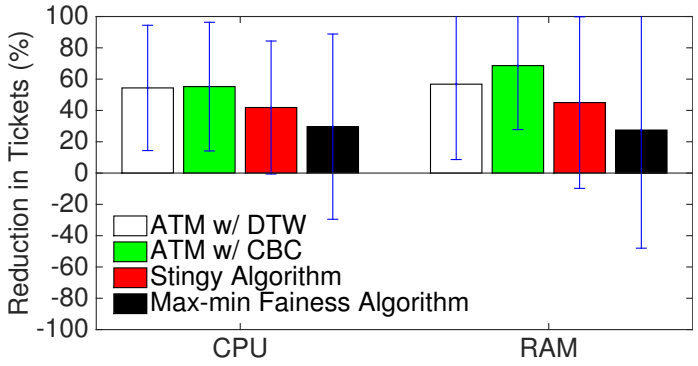


Figure 6.10: Comparing ticket reduction: ATM, max-min fairness, and stingy resizing algorithms.

small VMs while dissatisfying big VMs, which results in more ticket violations than the other policies. Another fact worth mentioning is that both versions of ATM are able to achieve higher RAM ticket reductions, due higher RAM provisioning compared to CPU.

6.5.2 ATM on a MediaWiki Cluster

We experimentally evaluate our ticket reduction techniques also on a cluster running MediaWiki, a latency-sensitive 3-tier web application composed by Apache (v2.4.7) as the application server frontend, memcached (v1,4.14) as in-memory key-value store, and MySQL (v5.5.40) as the database backend. The testbed is composed of four identical physical servers. Each server runs Ubuntu server 14.04 LTS and is equipped with 16 GiB of DDR3 RAM with up to 41.6 GiB/s bandwidth, a 4-core Intel Core i7 3820 processor @ 3.6 GHz with SMT, one 2-TB Sata III 7200 rpm hard disk, and one Gigabit Ethernet adapter. Three servers host the VMs using QEMU-KVM (QEMU v2.0 with KVM on Linux kernel 3.13) as hypervisor. Each VM comprises two virtual CPUs and 4 GiB of RAM. The forth server is used as the experiment orchestrator and load generator. Each application tier is deployed into a separate VM. We consider a scenario of hosting two MediaWiki applications on these 4 physical servers, termed as wiki-one and wiki-two, see Figure 6.11. For

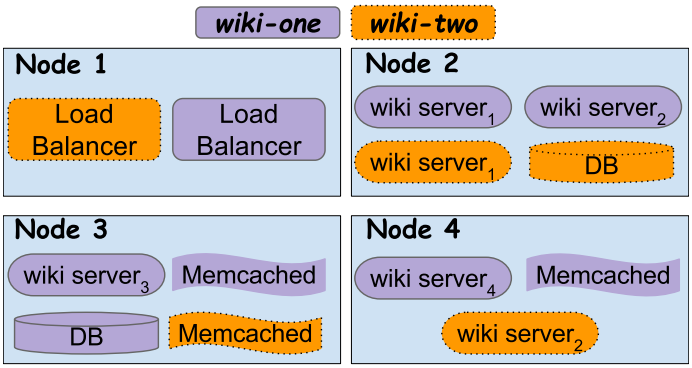


Figure 6.11: MediaWiki testbed.

wiki-one, there are 4 Apache servers, 2 Memcached, and 1 DB, whereas there are only 2 Apache, 1 Memcached, and 1 DB in wiki-two. For each wiki, we have one load balancer that distributes the requests across the different apache front-ends. The workload generator creates requests alternating between low and high intensity periods, each lasting one hour.

Figure 6.12 illustrates the CPU usage series across all VMs located on nodes 2, 3 and 4 against the ticketing threshold set to 60%. The figure shows the CPU usage levels without and with ATM resizing. One can observe that indeed resizing is very effective in achieving CPU usage levels across time and all VMs below the 60% threshold. The consequent ticket reduction is dramatic: tickets drop from 49 to only 1.

Besides ticket reduction, we also show performance values for the two wiki applications, see Figure 6.13. The figure plots the the average user latencies (response times) and average throughput (the average number of successful served requests per unit of time). For wiki-one, the mean response times with resizing decrease from 20% (from 582 ms to 454 ms) comparing to the original experiment, whereas throughputs are maintained at almost the same levels. For wiki-two, throughput increases by more than 20% (from 14 to 17 requests/sec) while response time increases by 7% (from 915 ms to 979 ms). This suggests

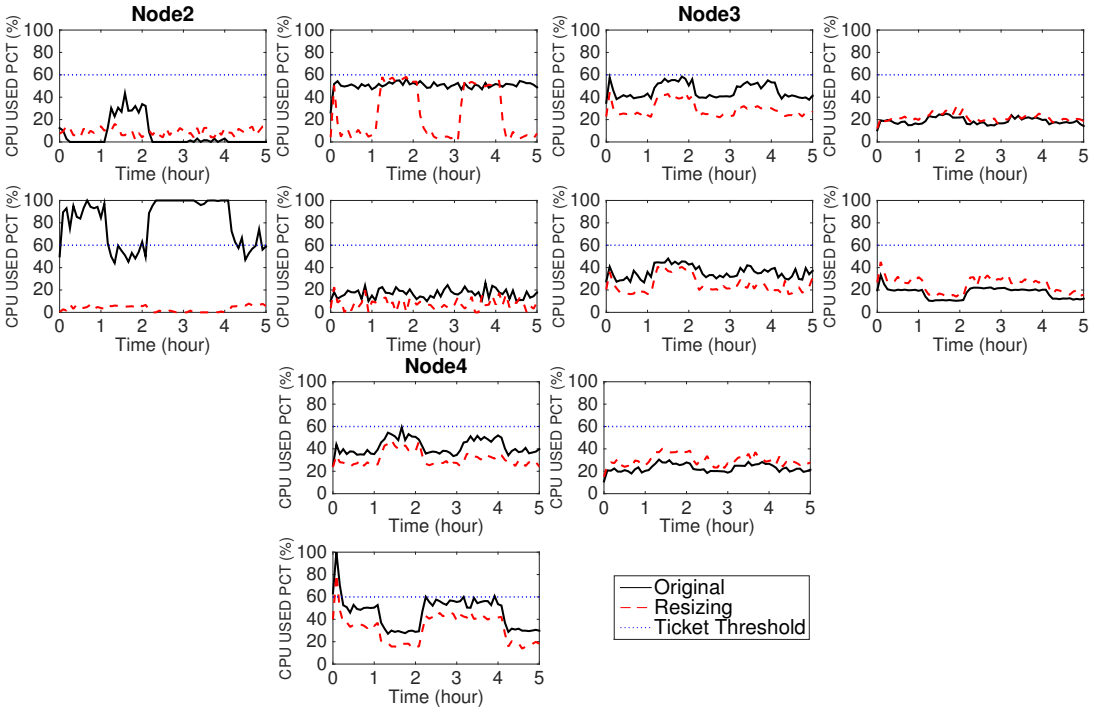
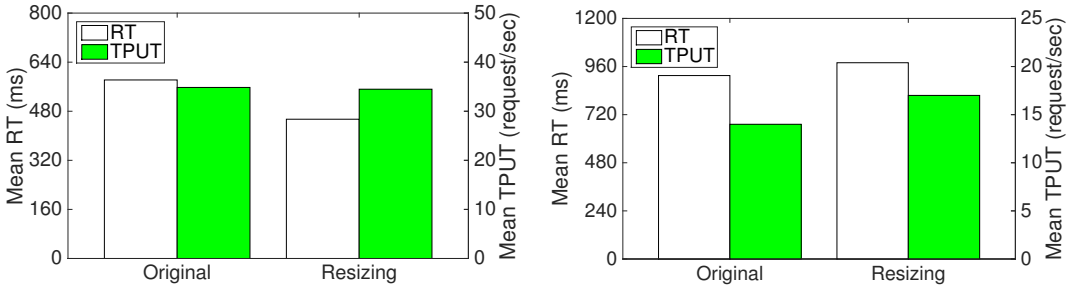


Figure 6.12: Overtime plots of CPU utilization for VMs located on Node 2, 3 and 4, with and without resizing.

that with ATM, the servers can fully serve the offered load, meet good performance values, while at the same time keeping the number of tickets to a minimum demonstrating the ultimate goal of ATM.

6.6 Chapter Summary

We presented ATM, a methodology to achieve efficient VM resizing so as to reduce VM usage tickets that are issued in production data centers. We have shown the effectiveness of ATM in predicting usage series in production data centers by exploiting spatial usage patterns of co-located VMs within the same box and by using detailed prediction of a small subset of the usage series, allowing the methodology to scale well. This prediction drives



(a) wiki-one (b) wiki-two

Figure 6.13: Performance comparison for wiki-one and wiki-two: original and resized with ATM.

the development of a VM resizing policy that is shown effective on a production trace and a working prototype.

Chapter 7

Reduction of Usage Tail Violations

Chapter 6 proposes an active resizing policy among co-located VMs in order to reduce the usage tickets in VMs. But this method does not reduce tickets which come from resource usage violations in physical boxes. In this chapter, we focus on reducing the usage tail violations in physical boxes in cloud data centers. We continue with the analysis of the IBM data center trace presented in Chapter 6. Recall that the trace data correspond to 6K physical boxes serving more than 80K VMs over a time period of a week. Our analysis shows that anomaly instances (AIs) fall into two categories: single AI where the duration of the anomaly is short and continuous AI where the duration of the anomaly is long. Figure 7.1 gives an overview of usage anomaly instances. Resource usage is typically reported within discrete time windows (e.g., in our trace each time window equals to 15 minutes). While the usage series has fluctuations across time, it goes beyond the target value three times. Twice the usage exceeds the target value for a short time of a single time window (single AI). Once the usage exceeds the target value for a long time, corresponding to multiple consecutive time windows (continuous AI). The trace characterization points to one more important factor that distinguishes single and continuous AIs: we find that not only the *duration* but also the *magnitude* of a continuous AI is larger than the single AI.

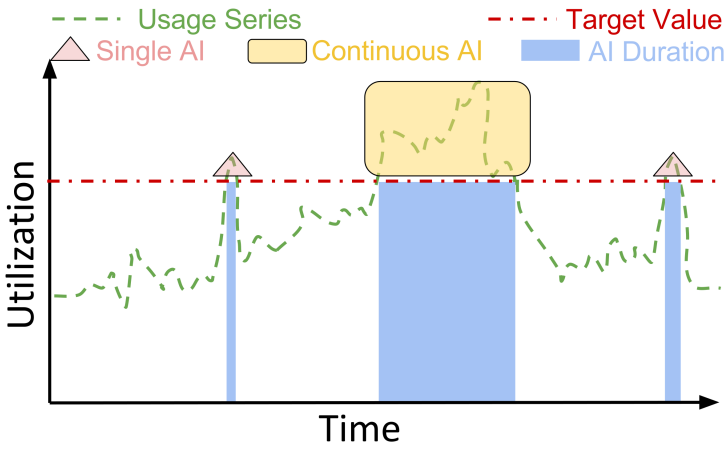


Figure 7.1: How to determine a usage anomaly instance. An example on CPU usage.

While single AIs may be considered relatively harmless, continuous AIs have the potential to significantly undermine the user perceived performance.

In this chapter, we develop a tail-driven anomaly avoidance policy, termed *TailGuard*, which aims to eliminate or at least drastically reduce continuous AIs in physical servers. For example, *TailGuard* tries to ensure that the box CPU usage is below the predefined target (e.g., 60%) for 95 percent of the time. To motivate the design of *TailGuard*, we first do a detailed, post-hoc workload characterization study of usage time series (for both CPU and RAM) in production data centers. *TailGuard* particularly consists of two steps: a light-weight tail usage prediction method that explores the power of vast number of last values of usages and a VM cloning strategy that redistributes the box CPU and RAM loads based on tail prediction. Overall, this work makes three main contributions: AI characterization, usage tail prediction, and anomaly mitigation.

This characterization analysis allows us to view the statistical characteristics of usage time series and focuses on the properties of their tails. The key findings are as follows: 1) the culprit of continuous CPU AIs is VM consolidation. 2) CPU tail usage is highly correlated to the mean CPU usage of physical servers and follows a Normal distribution. 3)

RAM anomalies do not relate strongly to VM consolidation in contrast to CPU anomalies.

Based on these observations, the proposed *TailGuard* avoids AI, particularly CPU, by predicting the tail usage of box CPU and redistributing VMs across boxes in an online fashion. *TailGuard* first predicts the box CPU tail usage, by capturing the steady state of tail distribution with respect to different levels of mean usage. In contrast to conventional time series prediction, *TailGuard* is not only light-weight since it uses very recent data, e.g., past day, instead of long history of usage series, but also aware of the resource availability of the tenants. Secondly, based on tail predictions and the level of resource availability for each tenant, *TailGuard* redistributes box loads by proactively creating and placing VM clones so as to ensure the box tail usage does not exceed the target value. The VM cloning strategy combines the advantage of VM migration and load balancing at the cost of duplicating the VM memory footprint.

TailGuard is evaluated using trace driven simulation. Results are summarized as follows: 1) the proposed prediction method of tail usage is computationally much cheaper compared to accurate but expensive time-series predictions (e.g., neural networks), while balancing tail usages across boxes; 2) VM cloning achieves a ten percent higher reduction in CPU tail target violations than classic VM migration. It is also noteworthy that, thanks to the reactive load balancing, our method achieves CPU tail usage reduction with minimal RAM usage violation increase (up to 3 percent), which is lower than the one achieved by proactive VM migration. Even by allowing target violations in up to 5 percent of the time windows, *TailGuard* not only reduces the number of CPU AIs by 60 percent, but also mitigates the duration of continuous CPU AIs dramatically, from a maximum duration of over 25 time windows to 2 time windows.

The outline of this work is as follows: Section 7.1 discusses related work. Section 7.2 provides a characterization study of the CPU and RAM AIs. We propose *TailGuard*,

describing tail usage prediction method and a VM cloning strategy in Section 7.3. An extensive evaluation of *TailGuard* for CPU AI reduction on production traces is discussed in Section 7.4. Section 7.5 concludes this chapter.

7.1 Related Work

Performance anomaly detection has been the subject of many workload characterization studies in recent years to improve system reliability by better understanding the reasons behind system failures and/or bugs [15, 92, 28, 97]. These studies focus on statistical analysis of trace data ranging from production data centers to large-scale storage systems. Online performance anomaly detection methods have been proposed [29, 26, 52] in order to detect anomalies in the upcoming future using either statistical methods or machine learning techniques. An online performance anomaly prevention method for cloud computing is proposed in [105], which integrates online anomaly prediction and learning-based cause inference with predictive prevention actuation. The spatial-temporal dependencies in usage time-series are explored in [118] for VM resizing, with an objective to reduce all VM performance tickets in data centers, i.e., all usages need to be below the target. In contrast, *TailGuard* focuses on avoiding box tail violations and makes a conscious tradeoff between resource requirement and anomaly avoidance, based only on the last values of usages collected from a vast number of boxes.

VM migration on a cluster of physical servers has long been proposed to mitigate performance anomalies via load balancing [27]. The main question is determining *when* and *how* to migrate VMs, aiming to optimize resource usage (e.g., network bandwidth [104]) and performance measures, including non-traditional ones such as energy consumption [47]. Several migration algorithms have been proposed including *best-effort* online VM migration [70, 104] as well as several performance models of online VM migration [71, 74].

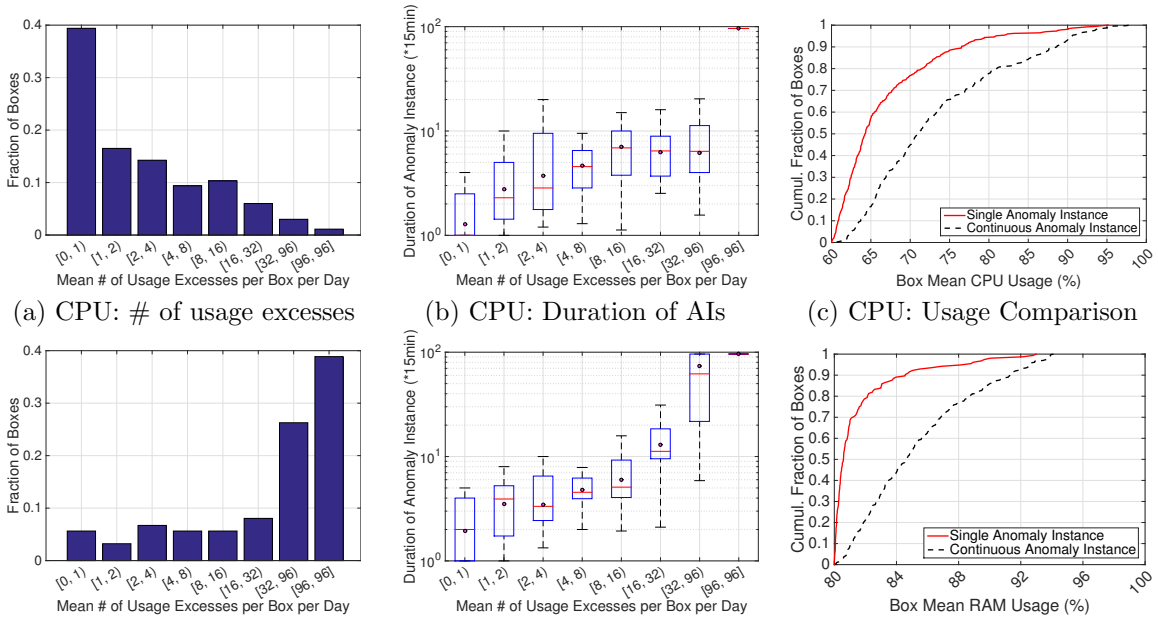
Yet, the performance effect of migration overhead cannot be disregarded [12], especially in systems where service availability and responsiveness are under Service Level Agreements (SLAs). Different from VM migration, in this chapter, *TailGuard* makes use of VM cloning to handle excess load, opportunistic job placement, and parallel processing [61]. VM cloning enables VM management that is more *flexible* and of *finer-granularity* control than VM migration.

7.2 Characterization

The trace considered here comes from production data centers serving various industries, including banking, pharmaceutical, IT, consulting, and retail, and contains CPU and RAM utilization at a time granularity of 15 minutes for 6K physical boxes hosting more than 80K VMs during a 7-day period from April 3, 2015 to April 9, 2015. Naturally, the level of consolidation is very high, i.e., on average 10 VMs are consolidated within a single physical box [16].

7.2.1 Overview

We first look into daily statistics of anomaly instances, assuming that an anomaly instance is triggered when a resource usage exceeds a *target* [118]: 60% for CPU and 80% for RAM. Figure 7.2(a) illustrates the empirical PDF of the daily average number of CPU excesses per physical box. Note that each excess corresponds to the mean utilization across 15-minute time window being above the target value. 40% of the boxes have a daily average number of excesses below one. After that the fraction of boxes rapidly decays with only 3% of boxes which experience over 32 excesses per day (i.e., for more than 8 hours per day). Figure 7.2(b) focuses on whether these excesses are single or continuous AIs. The boxplots in this figure show the 25^{th}, 50^{th}, and 75^{th} percentiles of the anomaly duration, the



(a) CPU: # of usage excesses (b) CPU: Duration of AIs (c) CPU: Usage Comparison

(d) RAM: # of usage excesses (e) RAM: Duration of AIs (f) RAM: Usage Comparison

Figure 7.2: CPU and RAM: overview of anomaly instances.

whiskers correspond to extremes of the distribution, and the dots represent the average. Note that the y-axis is in logscale and in units of 15-minute time windows. The figure clearly illustrates that most anomaly instances are continuous, i.e., longer than one time window. Figure 7.2(c) illustrates the relationship between the box mean CPU usage across the excesses and the type of anomaly that it experiences. Here, the CDFs of single and continuous anomaly instances are presented: it is clear that continuous anomaly instances have higher usages than single ones.

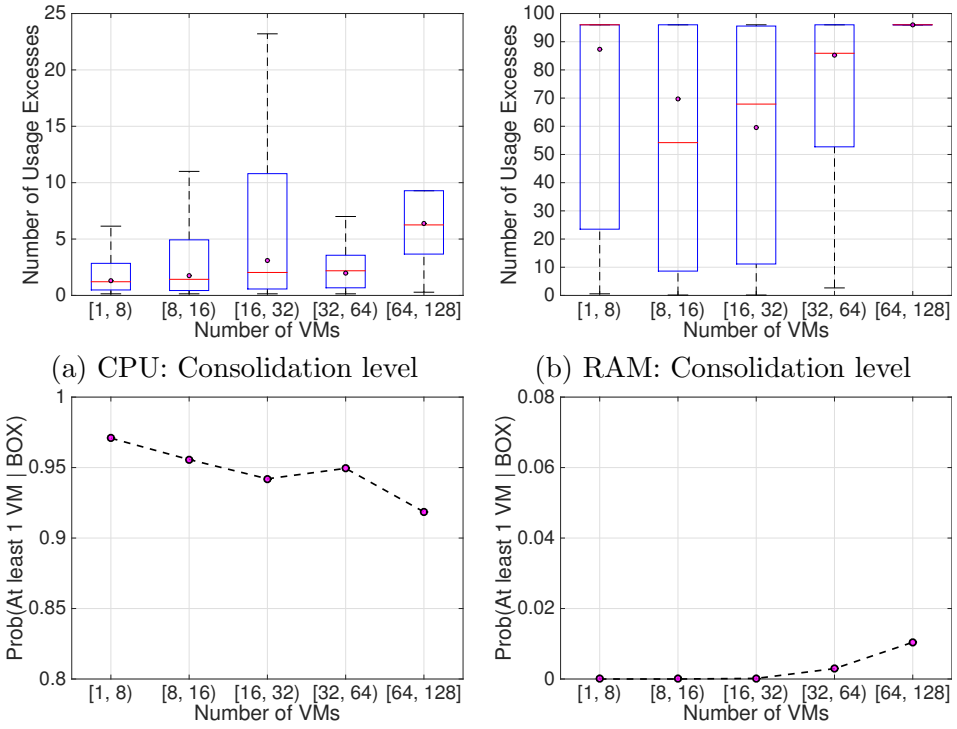
Figure 7.2(d)-(f) presents similar results as Figure 7.2(a)-(c) but for RAM. Figure 7.2(d) illustrates the empirical PDF of the average number of RAM excesses per box per day and shows that RAM excesses seldom come alone: from at least 32 (26% of boxes) up to 96 (40% of boxes). Consequently, the boxplots of Figure 7.2(e) illustrate that these anomalies are mostly continuous. Finally, Figure 7.2(f) shows that the CDF of single and continuous

anomaly instances for different box mean RAM usages across the anomaly instances. RAM continuous anomaly instances have higher usages than single ones and the difference is even larger comparing to CPU anomalies. In summary, Figure 7.2 illustrates that for both CPU and RAM anomaly instances, it is the continuous ones that tend to exceed significantly the target value. Hence, continuous anomalies have the potential to harm performance for a long period of time. This motivates us to look in detail into the tail usages for CPU and RAM.

7.2.2 Root Cause Analysis for Box Anomaly Instance

A natural question is whether there is any relationship between the VM consolidation level and the number of CPU/RAM anomaly instances. Figure 7.3(a) presents boxplots that show the 25^{th}, 50^{th}, and 75^{th} percentiles (boxes), the extremes of the distributions (whiskers) and the means (dots) of the number of CPU usage excesses for different ranges of VM consolidation levels on the x-axis. Consolidation level responds to the number of collocated VMs. Figure 7.3(b) presents the same information but for RAM anomaly instances. It is clear that higher VM consolidation levels result in more CPU anomaly instances while RAM anomalies do not show a strong relationship with the VM consolidation level.

Figures 7.3(c) and (d) illustrate the probability that at least one excess is observed on a VM residing on a given box that is also observed an excess for CPU and RAM usage, respectively. Note that these probabilities are reported as a function of the VM consolidation level. The figures illustrate that there is a strong relationship among the box anomaly instances and a VM anomaly instance for CPU, as shown by probabilities higher than 0.9 for all the consolidation levels. For RAM, the probabilities are less than 0.02 for all consolidation levels. The figures illustrate that there is a clear relationship among the hosted VM anomalies and the box anomalies for CPU; this relationship is clearly not



(a) CPU: Consolidation level

(b) RAM: Consolidation level

(c) CPU: Prob(VM|Box)

(d) RAM: Prob(VM|Box)

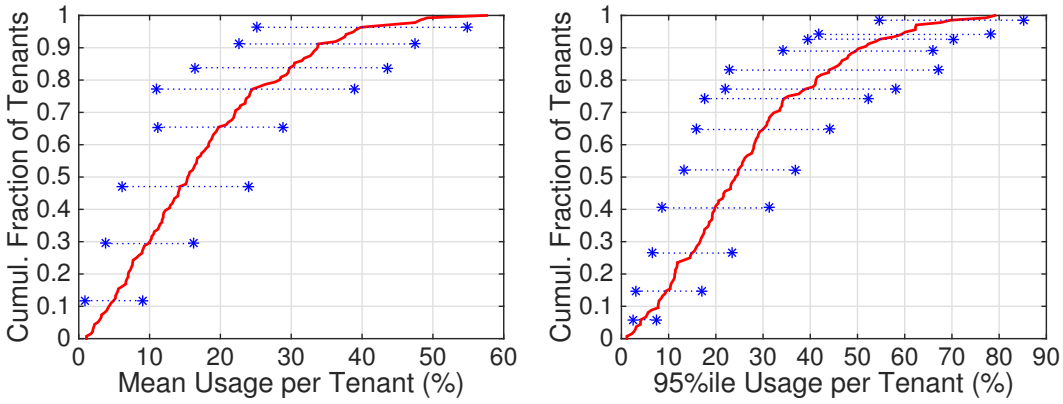
Figure 7.3: CPU and RAM: root cause analysis for the box anomaly instance.

present for RAM.

To summarize, the above two figures illustrate that CPU anomaly instances in boxes strongly depend on the VM CPU usage within these boxes, while this is not the case for RAM anomaly instances which are triggered mainly because of boxes themselves and not necessarily their residing VMs. This motivates us to focus on CPU AIs rather than RAM ones.

7.2.3 Is CPU Usage Balanced?

In a private cloud data center, which is a single *tenant* environment, the hardware, storage and network are dedicated to a single client or company. A *tenant* can be seen as a

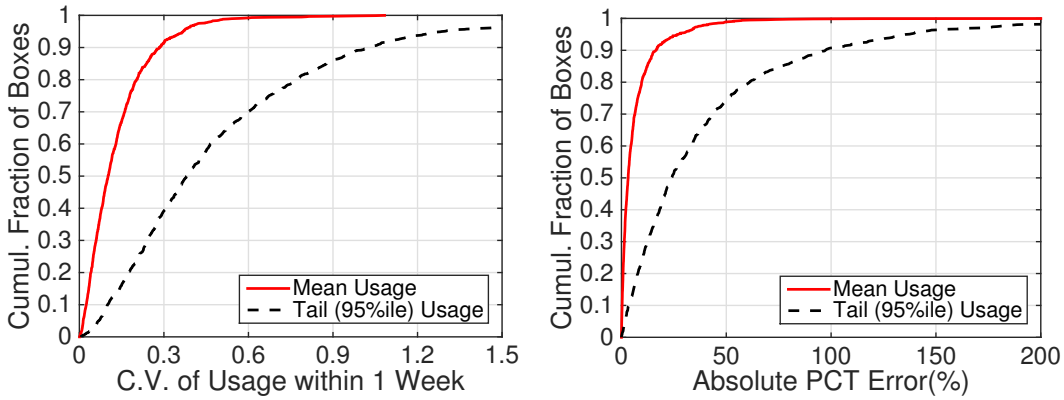


(a) Mean (b) Tail

Figure 7.4: CDF of box CPU usage per tenant: mean and tail.

cluster of boxes, which allows customized placement of VMs and assignment of resources. Figure 7.4(a) plots the CDF of box mean CPU usage across tenants. For selected means, we also plot the standard deviation. The figure shows that the average usage of boxes for the majority of tenants is low: 90% of tenants have an average CPU usage less than 35%. Yet, the standard deviation is large, suggesting that there is significant load imbalance across boxes. The same imbalance is observed for the CPU tail usages as shown in Figure 7.4(b) that plots the CDF of the 95%ile of box CPU usage across tenants. If we are able to predict the tail usages, then based on this information we can devise a balancing algorithm that reduces tail usage and consequently anomaly instances.

A simple common approach for predicting moments of time series makes use of the last value prediction, which predicts the future using the most recent observations. In Figure 7.5, we demonstrate that the last value prediction works well for predicting the mean usage, but not for the tail based on their coefficient of variation (C.V.), which is equal to standard deviation divided by mean. The C.V. allows us to combine the information on the mean and standard deviation into a single value. Figure 7.5(a) shows the CDF of the C.V. of the daily box mean and 95%ile usage computed over one week across all



(a) C.V. (b) Prediction Error

Figure 7.5: Predict box CPU mean and tail usages using the most recent day's observation.

boxes. It is clear that the box tail usages show higher C.V. values, on average 0.5, than the box mean usages, on average 0.12, indicating that the mean usages remain more constant over time. This is why applying simple last value prediction results in low prediction errors for the mean usages and high prediction errors for the tail usages. This is clearly shown in Figure 7.5(b) reporting the CDF of absolute percentage error (APE)[1] of the predictions. This observation motivates us to look into the relationship between the mean (rather constant) and tail (highly variable) usages, in order to predict the tail usage via the mean usage that is already predicted accurately using the last value.

To achieve this goal we divide the boxes into bins based on their mean CPU usage and compute within each bin the distribution of the CPU tail usages across all boxes falling into the same bin. Figure 7.6 illustrates the resulting PDFs for the 95%ile of CPU usages with a bin width of $\pm 3\%$. The figure shows that for each bin the tail usage distribution resembles a Normal distribution. For example, for the bin corresponding to a mean CPU usage equal to $(60 \pm 3)\%$, we see that average and standard deviation of a fitted Normal distribution of the 95%ile CPU usages are 79.9% and 8.2%, respectively. This allows us to

[1] $APE = \frac{|Actual - Prediction|}{Actual}$

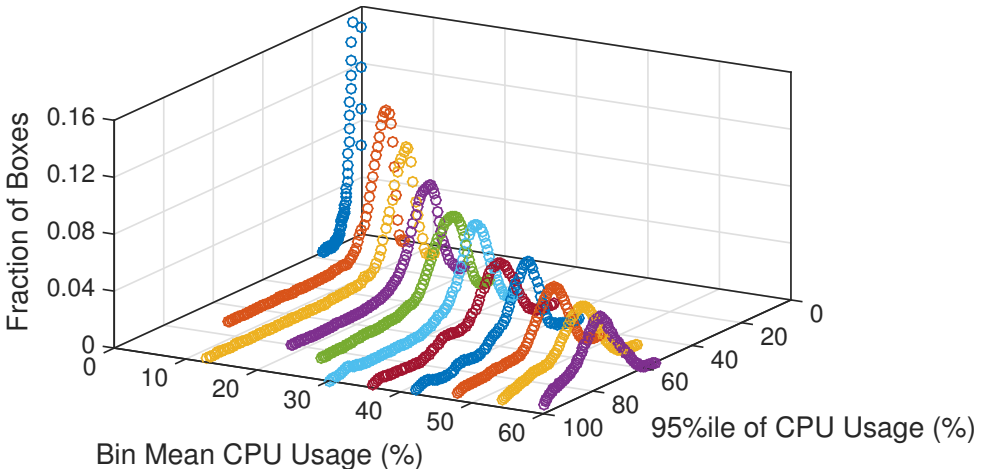


Figure 7.6: PDF of box tail usages for different mean usage bins.

propose an anomaly instance reduction method based on the mean usage and the (already known) distribution of the tail usages that correspond to this mean usage level.

7.3 *TailGuard* Policy

In this section, we present *TailGuard*, a tail-driven anomaly avoidance policy, that aims to enhance datacenter tenants' dependability by continuously ensuring their box CPU tail usages are below a predefined target value. To avoid suffering from continuous AIs and over-reacting to spontaneous single CPU AIs, *TailGuard* focuses on bounding the CPU tail usages for each tenant. *TailGuard* proactively manages the CPU and RAM usages of the boxes by intelligently distributing their load, i.e., learning from the past, predicting the future, and actuating at the present. *TailGuard* consists of two key steps: (i) CPU tail usage prediction and (ii) box load redistribution via VM cloning and online monitoring for workload management. The workflow of the proposed approach for box CPU tail usage reduction is shown in Figure 7.7. The focus of the tail prediction module is three-fold: prediction accuracy, computational efficiency, and awareness of the overall resource

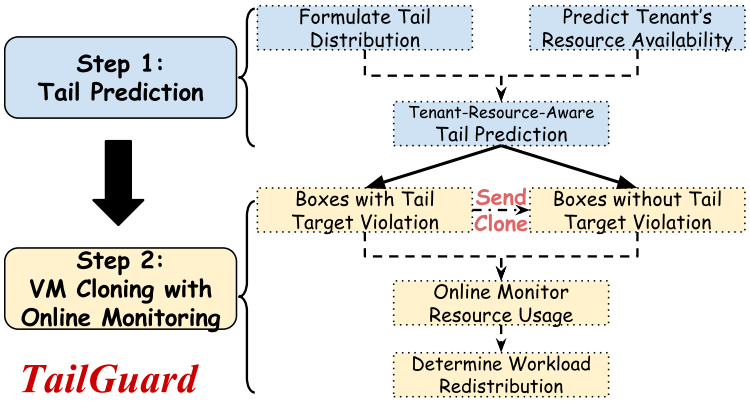


Figure 7.7: Overview of *TailGuard* to avoid box CPU tail violation.

availability of tenants. To achieve good accuracy with low computation overhead and also low requirement of data of the immediate past of a single tenant, *TailGuard* leverages trace data across *all* tenants' boxes, by constructing the empirical distribution of tail usages. We advocate to use simple statistics related to such a distribution, e.g., the mean and standard deviation, to predict the future tail usage of each box. In addition, *TailGuard* determines the prediction scheme, i.e., the specific statistics related to the distribution of tail usages, depending on the resource availability of each tenant.

The tail prediction is used as input for redistributing CPU and RAM loads across boxes in each tenant such that their box CPU AIs, particularly the continuous ones, are mitigated and avoided, with no or only few additional RAM usage violations happening. We propose a tail redistribution policy via VM cloning, essentially by combining the ideas of migration and load dispatching. Based on the CPU tail usage prediction for all boxes belonging to a tenant, *TailGuard* first proactively creates VM clones for boxes that need to reduce their loads, and places clones on boxes with spare capacity. *TailGuard* then dispatches the CPU loads across original and cloned VMs, while assuming memory loads are replicated on both original and cloned ones. The overview of the proposed VM cloning strategy is presented

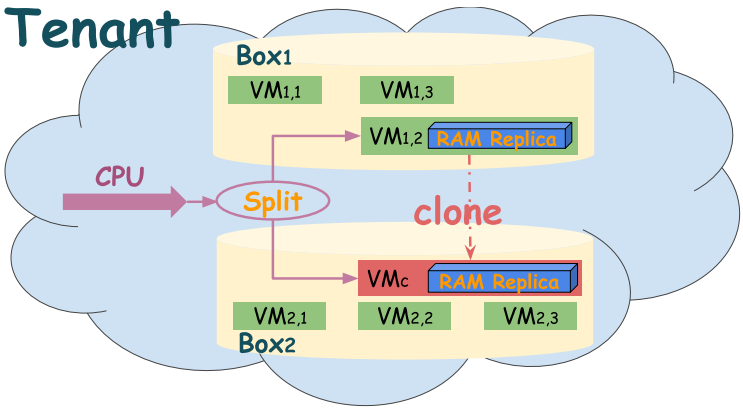


Figure 7.8: Overview of VM cloning and workload distribution.

in Figure 7.8.

After VM cloning, *TailGuard* enables online monitoring of CPU and RAM usage for each box. When unexpected CPU or RAM usages are detected in the boxes with VM clones, *TailGuard* reactively redistributes the workloads between the cloned and original VMs, such that the boxes with VM clones stay with as few CPU AIs or RAM violations as possible.

7.3.1 Predicting Box CPU Tail Usage

First we illustrate how *TailGuard* predicts individual box CPU tail usage, based on the distribution of tails observed from a large population of boxes. Our solution is motivated by the findings in Section 7.2: CPU tail usages are variable across time, while mean CPU usages remain constant, see Figure 7.5. In addition, CPU tail usages appear to follow Normal distributions after binning boxes based on their mean usages as reported in Figure 7.6. We incorporate the resource availability of each tenant, as an indicator that determines whether the tail prediction should be aggressive or conservative, i.e., the best safety margin to use.

Our objective is to predict the tail usage, L, for a given box by the mean and standard deviation of the tail distribution it belongs to. As such, we can write

$$L = \overline{T} + \alpha S_T \tag{7.1}$$

where $\overline{T}$ and S_T denote the mean and standard deviation of the tail distribution of the box, and α is a safety margin for the tail prediction based on the resource availability of the tenant. For tenants with abundant resources, higher α values allow to over-estimate the VM resources. This allows the use of more spare resources from boxes without tail target violations and a stronger tail target violation avoidance. For tenants with scarce resources, lower α values allow for more conservative estimates of the VM resources. This results in less requests of spare capacity from boxes without tail target violations protecting such boxes from potential tail target violations. We first explain how to obtain the box tail distributions by binning, and then extract the mean and standard deviation of the tail distribution per bin, to finally derive the tenant-resource-aware (TRA) tail prediction based on the properties of tail distribution.

7.3.1.1 Finding the Tail Distribution

Here, we describe how to bin boxes by their mean usages such that we can obtain $\overline{T}$ and S_T (of the tail distributions) as a function of their mean usages. Figure 7.6 shows how binning tail usages from all boxes by their mean usages can result into empirical distributions that resemble the Normal distribution. We experiment with different bin widths of mean usages, e.g., $\pm 1\%$, $\pm 2\%$, $\pm 3\%$ (as used in Figure 7.6) or higher. Different bin widths result in different number of samples in each bin. On the one hand our objective is to find a bin width that is big enough to contain a sufficient number of samples in each bin to be statistically significant, such that the box tails in the majority of bins follow a Normal

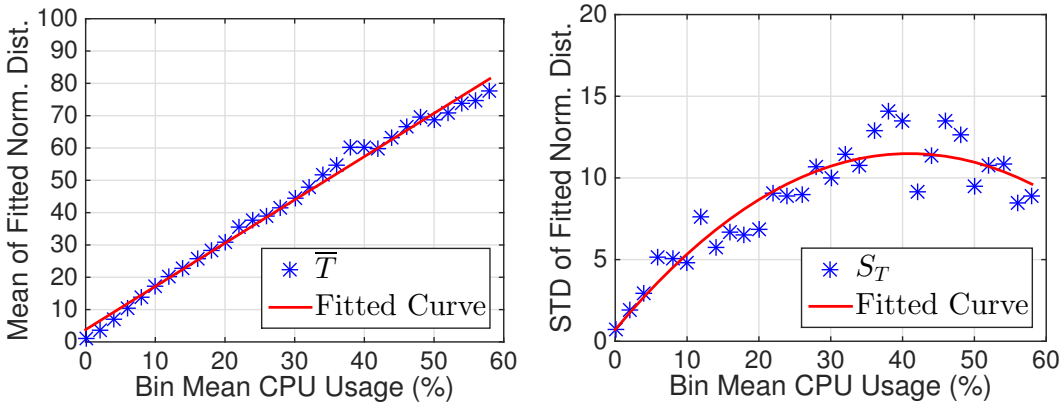
distribution. On the other hand we want the bin width to be small to have a stronger relationship between the mean and tail usages. Specifically, we compute a pair (M,T) for each box and for *all* tenants, where M represents the mean usage and T is the tail usage, and we bin them by their M values. From the data set considered in this work, we empirically conclude that using a bin width of $\pm 1\%$ is already sufficient such that all bins contain at least 30 tail samples [88], and more than 90% of the bins follow a Normal distribution as verified by the Kolmogorov-Smirnov test [78].

We then use the tail binning results to answer the following question: can we find a compact representation to describe the relationship among the bins, regarding to statistics of tail usages. For each bin, we extract two sets of pairs: (i) $(\overline{M},\overline{T})$ – the average of M, i.e., the bin mean, and average of T, and (ii) $(\overline{M},S_T)$ – the average of M and standard deviation of T. We plot these two sets of pairs for all bins in Figure 7.9(a) and (b), respectively. Visual inspection of Figure 7.9 indicates that there exists a linear dependency between the bin mean and the average tail usage across bins and a quadratic dependency between the bin mean and the standard deviation of tails across bins. Given $\overline{T}$ and S_T for each bin mean $\overline{M}$, we fit the two curves using Eq. 7.2-7.3. Here a_0 and a_1 are the coefficients of the linear fitting for $\overline{T}$, whereas b_0, b_1, and b_2 are the coefficients of the quadratic fitting for S_T.

$$\overline{T} = f_1(\overline{M}) = a_0 + a_1(\overline{M}) \tag{7.2}$$

$$S_T = f_2(\overline{M}) = b_0 + b_1(\overline{M}) + b_2(\overline{M})^2 \tag{7.3}$$

Substituting Eq. 7.2-7.3 into Eq. 7.1, one can thus obtain the closed-form expression for the estimates of the tail usages as a function of its mean usages. Essentially, to predict the CPU tail usage for a box, we need to "learn" the fitting coefficients of functions Eq. 7.2-7.3



(a) Fitting of $\overline{T}$ (b) Fitting of S_T

Figure 7.9: Fit mean and standard deviation of tail (95%ile) distributions for different bins, using bin mean usages.

from the empirical distributons of historical data and then use simple last value prediction of $\overline{M}$, as argued in Section 7.2.3. The final step is to decide the value of α in Eq. 7.1, which is addressed in the next subsection in the light of the overall resource availability of tenants.

7.3.1.2 TRA Tail Prediction

When considering Eq 7.1, the most accurate prediction is achieved by setting $\alpha = 0$, i.e., $L = \overline{T}$, whereas higher values of α will tend to over-estimate L providing increasing safety margins. As one of the ultimate goals is to mitigate anomaly instances of tail usages exceeding the target value, we determine the α value for all boxes belonging to the same tenant, based on the tenant's resource availability. Low-CPU-utilized tenants have abundant resources for tail usage reductions, therefore *conservatively* predicting tail usages (i.e., higher α safety margins), and *aggressively* re-allocating resources reduce the tail usage below the target value, but do no harm to the original boxes with no tail target violations. High-CPU-utilized tenants have relatively scarce resources, therefore *aggressively* predicting tail

usages (i.e., lower α safety margins), and *conservatively* re-allocating resources result in less resource re-allocation for tail usage reduction, but guarantee that the original boxes without tail target violation remain in the 'safe zone'.

The above illustration suggests that α should be customized based on the CPU availability of tenants. We propose to compute α_i for tenant i by considering the target value TG equal to the average tail usage obtained under an optimal case, where all boxes of tenant i undergo perfect load balancing, i.e., every box is equally utilized at the optimal mean value, M_i^*. Basically, one can compute the maximum value of α_i by solving the following equation,

$$TG = f_1(M_i^*) + \alpha_i f_2(M_i^*). \tag{7.4}$$

where f_1 and f_2 represent the fitted functions of Eq. 7.2-7.3.

Additionally, we impose a lower bound on α_i, i.e., $\alpha_i \geq 0$, to ensure that the tail prediction is at least equal to $\overline{T}$. All in all, the TRA tail prediction $\hat{L}_j$ for box j belonging to tenant i is

$$\hat{L}_j = f_1(\hat{M}_j) + \alpha_i f_2(\hat{M}_j), \tag{7.5}$$

where $\hat{M}_j$ denotes the predicted mean usage for box j. Comparing the predicted tail, $\hat{L}_j$, with the target value indicates if box j is expected to have a tail target violation or not.

7.3.2 VM Cloning

Here, we explain how *TailGuard* redistributes the box CPU and RAM loads by VM cloning strategy, such that the probability of boxes in each tenant with CPU tail target violation is minimized. It consists of two steps: (i) proactively create VM clones on boxes at the beginning of the optimization horizon, e.g., one day ahead, and (ii) online monitor the resource usage and reactively redistribute loads among original and cloned VMs. The intention of additional VM clones is to redirect the load from the original box host to the box

host of the cloned VM at the cost of duplicated memory footprint. We redistribute incoming requests to the original and cloned VMs via simple Domain Name Server (DNS) based load balancing [79]. This scheme is simple and compatible with a large set of applications.

7.3.2.1 Creating VM Clones

The TRA tail prediction provides two key pieces of information for cloning VMs for each tenant i: the predicted tail usage for each box j, $\hat{L}_j$, and the optimal box mean usage under perfect load balancing, M_i^*. Based on the comparison of the target value and the predicted box tail usage, *TailGuard* decides which boxes need to migrate some workloads through additional VM clones, these boxes constitute the *reduction set*, and which boxes have spare capacity to receive VM clones on top of their existing VMs, so-called *increasing set*.

Creating VM clones: The VM configuration that can benefit most from cloning is the one with large CPU usage and low RAM usage, as cloning has the advantage of distributing CPU load but at the cost of replicating memory usage. Consequently, for each box in the *reduction set*, *TailGuard* ranks their VMs (indexed by k) by the VM's *cloning benefit* ratio, ρ_k, defined as the mean VM CPU usage C_k divided by the mean VM RAM usage R_k computed over all CPU AI points in the training window W, e.g., $W = 1$ day, $\rho_k = \frac{C_k}{R_k}$. The top ranked VMs have heavy CPU loads but low memory footprints.

For each box in the *reduction set* of tenant i, starting from the highest ranked VM, *TailGuard* clones VMs to reduce the box mean CPU usage from its the latest value to M_i^*. We aggressively assume that upon cloning of VM k, the box CPU usage can be reduced by C_k, except for the last cloned VM from this box. When cloning the last VM from box j, removing all the CPU workloads on this VM may reduce the box CPU mean usage to *less than* M_i^*. As a result, for the last VM to be cloned, we only redistribute part of the CPU workload to the cloned one, such that the mean CPU usage of box j is exactly reduced

to M_i^*. At each iteration we update the reduction set, as soon as the resulting box mean usage is equal to M_i^*.

Placing VM clones: *TailGuard* ranks the boxes in the *increasing set* of each tenant i by the amount of spare resources, i.e., CPU and RAM, in a descending order with priority first to CPU and then to RAM. The spare CPU is defined as the difference between the current CPU usage and M_i^*, while spare RAM refers to the difference between current RAM usage and RAM target value. When placing the VM clones, *TailGuard* starts from the top-ranked box regarding CPU and RAM spare resources, as long as it has sufficient amount of spare resources to cater to the demands of cloned VMs. Whenever the clone is placed we update the spare resources and the box rank in the increasing set. We terminate the cloning strategy when either the reduction set or the increasing set exhausts.

7.3.2.2 Maintaining Safety Margins

We advocate the use of online monitoring on allocating CPU loads across the original and cloned VMs and terminating clones upon observing RAM violation, to control that cloning does not inadvertently introduce more violations. The focus is to protect the performance of boxes that receive VM clones. *TailGuard* monitors online the CPU and RAM AIs for each box receiving VM clones. Upon detecting a box CPU tail target violation, e.g., accumulating more than 5 CPU excesses given a tail target metric of 95%ile and 96 observation points per day, we reduce the workload of the hosted cloned VMs proportionally to the intensity of the past observed excesses. On the contrary, if a box RAM violation is detected, we directly terminate the cloned VMs, based on the RAM usage rank in a descending order, till RAM usage is less than the RAM target value.

We note that the online workload distribution scheme proposed here can be implemented via DNS-based load balancing. When a user wants to send a request to a VM, it

has to first resolve the hostname to the IP address via a DNS lookup. DNS-based load balancing allows to associate multiple IP addresses to the same hostname so that different DNS lookups for the same hostname return different IP addresses. By updating the entries one can control the workload sent to the original and cloned VM. Since most applications use hostnames rather than IP addresses, DNS-based load balancing is compatible with a large set of applications.

7.3.3 Putting All Together

Lastly, we illustrate how *TailGuard* puts the proposed TRA tail prediction, and VM cloning strategy together. The optimization period is one day and the training period of the model is past W days. Past work [15] has shown that most VM migrations in corporate data centers occur once a day and around midnight. We propose implementation for VM cloning to follow this same time frame. Figure 7.10 illustrates the schematics of the proposed scheme. Particularly, *TailGuard* uses historical data of W days to learn the tail distribution, and derive the closed-form solution of tail estimates in Eq. 7.5. The higher the value of W is, the longer the historical data used. *TailGuard* then makes VM cloning decisions for the next day. Prior to moving into the next optimization period, *TailGuard* terminates all clones generated in the current period.

We experiment with prediction accuracy of the proposed scheme with two different lengths of W. Specifically, we present the CDF of absolute percentage errors to predict the 95%ile and 98%ile, using one and three days, see Figure 7.11. As the focus here is to see the impact of the training window length, we set all α_i to 0. We can see that there is no obvious difference in the CPU tail usage predictions between three-day and one-day training. Thanks to large amount of tail data collected from 6K boxes, one-day training already shows robust prediction results. We leverage many box observations

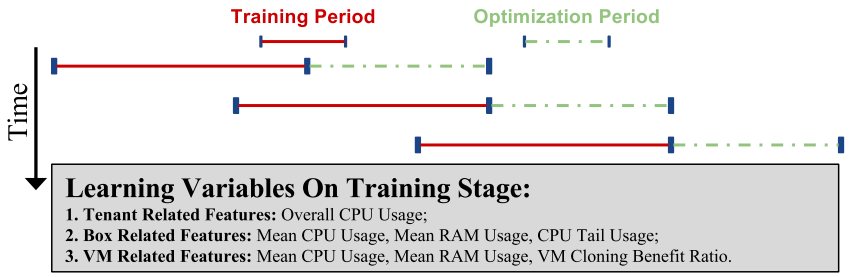
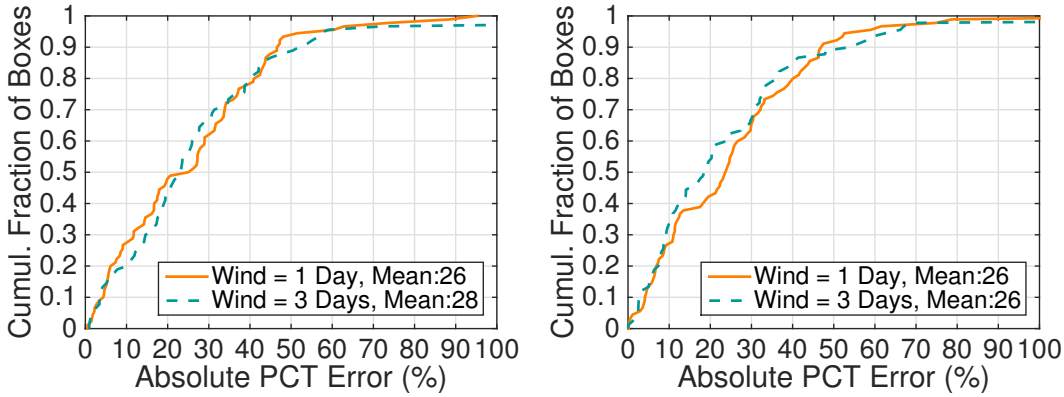


Figure 7.10: Dynamic method for tail prediction and VM cloning.



(a) 95%ile (b) 98%ile

Figure 7.11: CDF of box CPU tail usage prediction error for different training window sizes.

(spatial observations), rather than long-historical observations (temporal observations) for each box.

7.4 Evaluation

In this section, we evaluate *TailGuard* for mitigating performance anomalies for 80 data-center tenants, using more than 1K boxes and 10K VMs. Our focus is on the following metrics of interest: (1) reduced CPU tail target violations for boxes, (2) reduced CPU anomaly instances, and (3) reduced anomaly durations. Since accurate tail prediction is central to the effectiveness of the proposed strategy, we also compare the tail prediction

accuracy presented in Section 7.3 with the time series prediction based on neural networks that have been shown very effective on the same trace [120]. In the following, we first sketch the simulator design and present the effectiveness of the proposed methodology in reducing (continuous) AIs. We then highlight how two parts of *TailGuard*, i.e., the tail prediction scheme and VM cloning, outperform alternative approaches, i.e., time series prediction and VM migration.

7.4.1 Experimental Set Up

Simulator: We develop a trace-driven simulator that emulates the system dynamics. The input data of the simulator are CPU and RAM usages from VMs and boxes of 80 selected tenants. The simulator computes the box CPU/RAM usages as the sum of the hosted VM CPU/RAM usage plus the original box-only usage. When cloning a VM, its CPU usage is predicted and apportioned as described in Section 7.3. For RAM usage of each cloned VM, we use directly the VM value from the trace as we assume that RAM is replicated. For all statistics related to RAM usage we use the last value from historical data as prediction since the traces show that RAM usage for both boxes and VMs is stable across time.

Targets: Here, we consider the target usage values for box CPU and RAM for triggering tickets for anomaly instances at 60% and 80%, respectively. We experiment different tail usages, such that different tolerances of CPU anomaly instances are evaluated. This is customized to different system requirements. The specific CPU tail usages evaluated here are the 90^{th}, 95^{th}, and 98^{th} percentiles. In the following, we focus on presenting the reduction of CPU tail target violation per tenant, as well as the reduction in the number of CPU AIs.

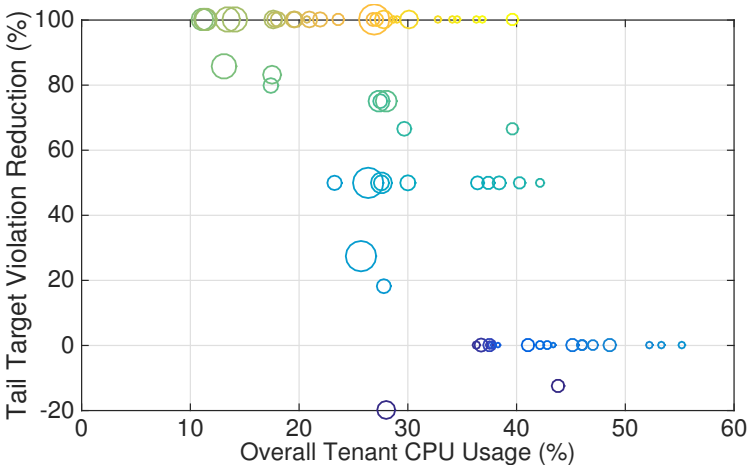


Figure 7.12: Reduction of CPU tail target violations for all tested tenants: the tail is set as 95%ile.

7.4.2 Big Picture

We first present an overview how *TailGuard* reduces tail target violations for the box CPU of the 80 tenants, see Figure 7.12. The usage tail considered here is 95%ile. Each point in Figure 7.12 represents a tenant, whose reduction of tail target violations is shown on the y-axis, while the average CPU usage is depicted on the x-axis. The number of boxes of each tenant is represented by the size of the bubble. A lower value on the x-axis shows higher spare resource availability. The figure clearly shows that for bigger tenants and for tenants that have higher spare capacity, the reduction is significantly higher due to the higher degrees of freedom in VM redistributing.

7.4.3 Effectiveness of Tenant-Resource-Aware Tail Prediction

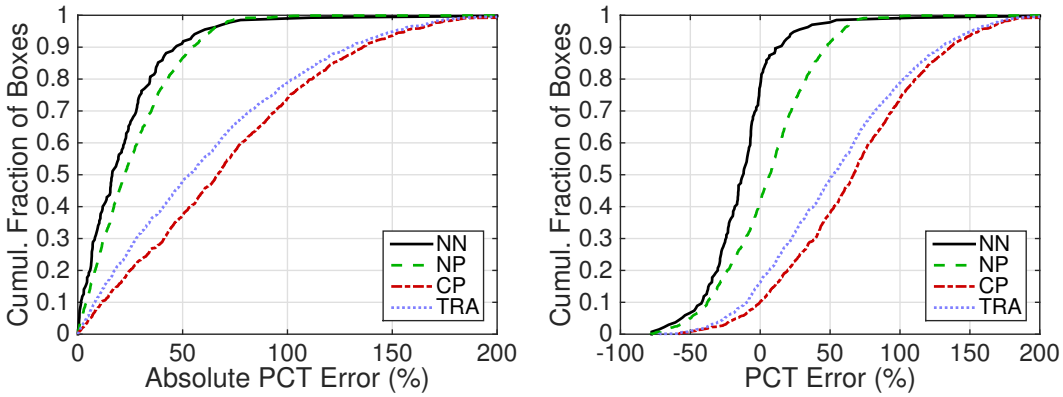
Here, we present how the proposed *TailGuard* can accurately capture the box CPU tail dynamics and effectively reduce the tail target violation when driving the VM cloning. We compare three variations for tail estimation with a neural network based time series prediction (`NN`) [120] that accurately forecasts the entire trajectories of usages in the trace.

In order to build NN prediction models, we first need to use historical data to train the model. We use the trace data that correspond to the past three days, which is sufficiently long to capture the time dependency within the series. We use the tail estimation scheme proposed in Section 7.3, i.e., $L = \overline{T} + \alpha S_t$ with different values of α:

- $\alpha = 0$, across all tenants, neutral prediction (NP), i.e., the mean of the tail distribution is used as the predicted tail;
- $\alpha = 2$, across all tenants, conservative prediction (CP), i.e., the predicted tail is the mean plus two standard deviations which corresponds to the 97.5^{th} percentile of the Normal distribution ;
- α_i, for tenant i, the proposed tenant-resource-aware tail prediction (TRA).

Prediction Accuracy: Figure 7.13(a) and (b) summarize the CDF of absolute and raw percentage of prediction errors for the box CPU tail for the three tested days. The two key findings are: (i) the proposed tail estimation scheme (of $\alpha = 0$) is almost as accurate as the time-consuming NN approach, and (ii) the proposed TRA overestimates the CPU tail, but is still less conservative than CP.

The lowest prediction errors are achieved by NN with average value of around 20%, but neural networks are computationally expensive and require long historical data for training. Given shorter training data and much lower computational complexity, NP can achieve very similar absolute prediction errors as NN. When looking to the distribution of raw errors, NP tends to overestimate the box CPU tails as more than 60% of errors are positive. In contrast, NN tends to underestimate the box CPU tail, since more than 70% of errors are negative. TRA and CP have significant higher absolute and raw errors, due to their conservativeness in estimation. In terms of raw errors, only 15% of prediction errors from TRA are negative. As the ultimate objective is to enable efficient resource management,



(a) Absolute PCT Error (%) (b) PCT Error (%)

Figure 7.13: CDF of CPU tail prediction errors using different methods: the tail is set as 95%ile.

conservative prediction tends to resource over-provisioning, which is more desirable than under-provisioning, which leads to high risk for performance anomalies.

Reduction of Tail Target Violation: To see the impact of the tail prediction schemes and to evaluate the proposed `TRA`, we use the three tail prediction schemes to drive VM cloning. Figure 7.14 summarizes the reduction of CPU tail target violations for each tenant, for different considered tail percentiles and prediction schemes. Each box in Figure 7.14 presents the distribution of tenants' reduction of CPU tail target violation, for different tail percentiles. Each rectangular box contains three horizontal lines that correspond to the 25^{th}, 50^{th} and 75^{th} percentiles, the circles show the mean. The whiskers show the extreme values in the distribution. We can see that the proposed `TRA` achieves the highest average reduction per tenant (shown by higher positions of circles), for all three tail percentiles. Specifically, the average reduction of CPU tail target violation under the `TRA` prediction scheme is around 50%, whereas `CP` and `NP` can only achieve average tail reduction per tenant at around $30\% - 40\%$. For all three tail prediction methods, the increase in number of RAM AIs is negligible, with all less than 3%. Another finding worth mentioning is that, when increasing the tail percentiles, i.e., from 90%ile to 98%ile, the

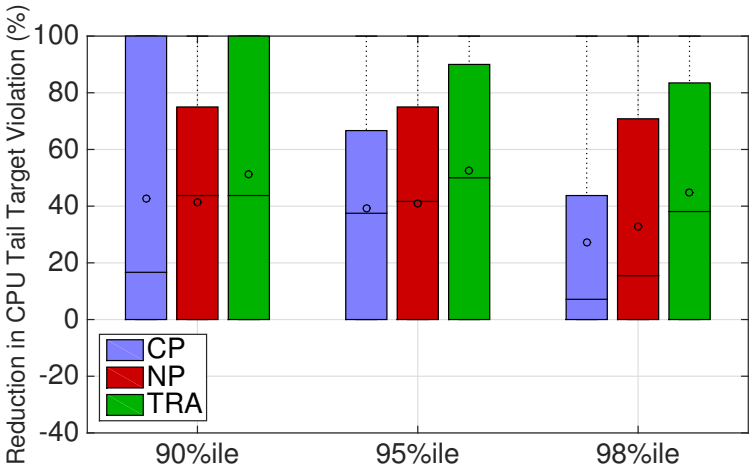


Figure 7.14: Comprison of CPU tail target violation reduction: `CP` *v.s.* `NP` *v.s.* `TRA`.

reduction drops slightly for all prediction schemes. This is because a more stringent performance requirement is applied, allowing only very few AIs, and the potential of reducing violation by redistributing the loads across boxes becomes lower. This also leads to the explanation why `CP` outperforms `NP` in case of 90%ile, but `NP` results into better reduction in the case of 95%ile and 98%ile.

To highlight the impact of different prediction schemes on mitigating tail target violations, we zoom into the performance of two tenants. One of the tenants has a lower CPU utilization, meaning high resource availability, whereas the other tenant has a higher CPU utilization. We list their reduction of CPU tail target violation in Table 7.1. `CP` is able to remove all tail target violations for the tenant with a higher resource availability but performs poorly for the second tenant. `NP` has the opposite performance for these two tenants, arguing for the need of a prediction scheme that can self-adapt to the resource availability. This is what `TRA` consistently does: tail prediction enables VM cloning to achieve the highest amount of reduction for both tenants, as it uses different α_i values for box tail predictions, based on the resource availability of different tenants.

Table 7.1: CPU tail target violation reduction: a case study comparing `CP`, `NP` and `TRA` with the tail set as 95%ile.

	Tenant with *higher* resource availability (mean CPU usage = 12%)	**Tenant with *lower* resource availability (mean CPU usage = 42%)**
`CP`	100	16.7
`NP`	75.0	50
`TRA`	100	50

7.4.4 Effectiveness of VM Cloning

Here, we highlight how *TailGuard* can reduce CPU tail target violation and AIs by VM clonning strategy, with negligible impact on the RAM violations. We compare the VM cloning with an alternative of VM migration only strategy, which migrates VMs from boxes to boxes, without replicating the memory. For a fair comparison, we provide the box tail prediction obtained from the `TRA` prediction scheme to both strategies. As both strategies aim to redistribute the box workloads, we use the same criteria to determine how many VMs to be moved out of certain boxes and how many additional VMs to be allocated to certain boxes, see Section 7.3.2.

CPU Tail Target Violation: Figure 7.15 summarizes the distribution of the reduction of CPU tail target violations per tenant, when applying VM cloning and migration on different tail percentiles. One can see that VM cloning is able to achieve a slightly higher reduction of tail target violations for CPU by roughly 10%, shown by the difference of mean values. Moreover, with VM cloning, the range of reduction of tail target violations for CPU across all tenants is much smaller, supported by the shorter box. In addition, cloning guarantees tail reduction while migration does not necessarily do so, see the whiskers in the respective boxes. Indeed, when the tail percentile is set to 98%ile, namely allowing only 2 CPU AIs for the entire day, VM migration can result into undesirable scenarios, i.e., certain tenants may experience a tremendous increment of CPU tail target violation as shown by the negative values of reduction, while VM cloning can ensure a more consistent

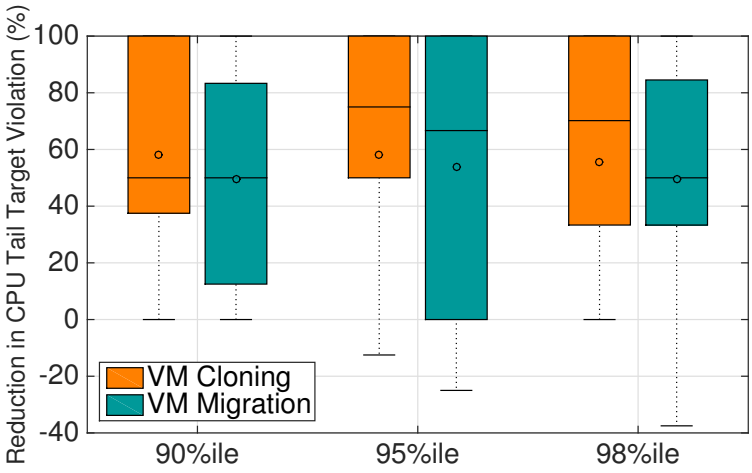
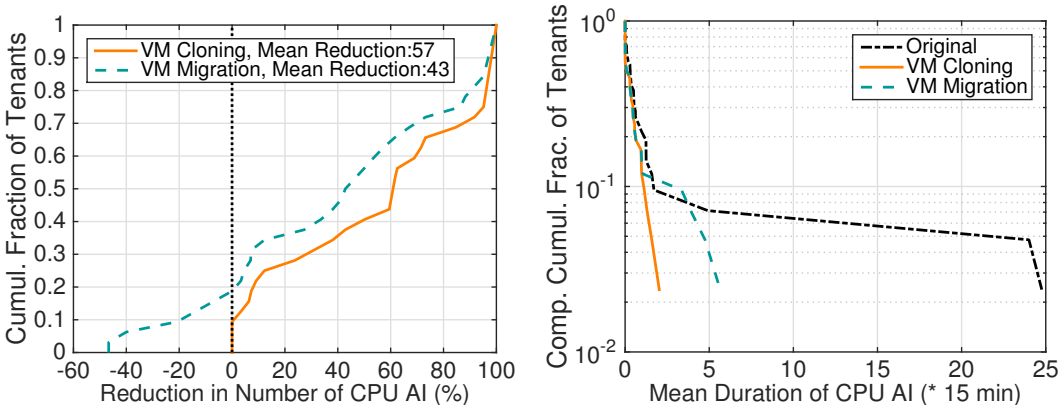


Figure 7.15: Comparison of CPU tail target violation reduction: VM cloning *v.s.* VM migration.

reduction in CPU tail target violations, and prevent the aggravation of CPU tail target violation by online usage monitoring and workload management.

Another advantage worth mentioning is that VM cloning ensures RAM performance, compared to VM migration. Our simulation results show that VM migration indeed results into significant increment of number of RAM AIs, with roughly 50% increment for each tenant on average. On the contrary, the proposed VM cloning strategy is able to bound the increment of RAM usage violation within 3%. This is thanks to the online usage monitoring and workload management, which is able to terminate clones upon detecting any RAM violation.

Occurrences and Duration of CPU AIs: Now, we present the difference between VM cloning and VM migration from the perspective of number of CPU AIs and their duration. Figure 7.16(a) and (b) present the CDF of average reduction in number of CPU AIs and the complementary CDF (CCDF) of average duration of CPU AIs per tenant, respectively. We could see that VM cloning can achieve a mean reduction of 57% in number of CPU AIs per tenant, while VM migration only reduces CPU AIs per tenant by



(a) Reduction in CPU AI (%) (b) Duration of CPU AI

Figure 7.16: Comparison of CPU AI for each tenant between VM cloning and VM migration: the tail is set as 95%ile.

around 43%. Moreover, similar to the case of reduction in CPU tail target violation, VM cloning guarantees almost no increment in the number of CPU AIs. However, there are 20% of tenants with an increased number of CPU AIs after VM migration. In terms of duration of CPU AIs summarized as CCDF in Figure 7.16(b), VM cloning can bound the duration of continuous CPU AIs below 2 time windows, whereas VM migration still results in 6 time windows of continuous CPU AIs in the worst case. We note that in the original testing trace, there are some tenants suffering from *long-term* continuous CPU AIs, as long as 25 time windows, i.e., CPU usage exceeds the target value of 60% for more than 6 hours.

7.5 Chapter Summary

In this chapter, we develop *TailGuard*, a tail-driven anomaly avoidance policy, which can effectively reduce continuous AIs for box CPU and avoid over-reacting to the spontaneous single AI. The design of *TailGuard* is based on a large-scale characterization study of production data centers. *TailGuard* is composed of two steps: a light-weight tail usage predictor simply leveraging the power of the vast amount of usage data across boxes,

and a VM cloning strategy combining the advantage of migrating VMs and balancing the loads between clones. Moreover, we incorporate the concept of resource availability per tenant into the tail prediction, such that *TailGuard* is able to adjust the conservativeness of prediction based on the flexibility of redistributing CPU loads. The extensive evaluation results on 80 tenants over 1K boxes over 3 days show that *TailGuard* is able to accurately predict the box tail usage, with an accuracy comparable to a neural network-based time series prediction. More importantly, while *TailGuard* purposely allows a small fraction of AIs to avoid excessive resource provisioning, it can still achieve a significant reduction in CPU AIs and drastically reduce the anomaly duration by 10 times.

Chapter 8

Future Work

In Chapter 6, we propose a spatial-temporal prediction model, which efficiently predicts the resource usage series in cloud data centers. Still, there exist several open questions, such as the trade-off analysis between spatial and temporal models and the selection method of signature series. In the future, we will work on the quantification of the trade-off between spatial and temporal models and propose different signature selection methods, in order to modify and customize the spatial-temporal prediction model.

Another interesting future direction is to further build on the models of Chapter 7, where we propose a cheap usage tail prediction method for physical boxes. Preliminary experiments show that *TailGuard* is very weak in predicting VM usage tails, this is because VM usage tails show much higher variations than boxes. Given this problem, we plan to develop a *state-aware* usage tail prediction method for VMs in the future.

When studying the traces from IBM cloud data centers, we find that a significant amount of usage series have the problem of *time holes*, i.e., for time periods there are no recorded trace data. This makes these traces unfit for prediction. We anticipate that the existence of spatial-temporal dependency of the usage series in data centers [118] offers us the opportunity to *intelligently* fill up these time holes, based on the remaining *correlated*

and *complete* series. In the following, we will further elaborate on these future work.

8.1 Spatial-Temporal Prediction Models

This project will focus on developing a *customized* and more *efficient* spatial-temporal prediction model. The proposed model in Chapter 6 is inflexible for user demands, i.e., it does not consider the trade-off between prediction cost and accuracy. Moreover, the previous prediction model only captures spatial characteristics to select the most representative signature series, but there is no clear consideration of temporal characteristics to build the signature series. To this end, we intend to:

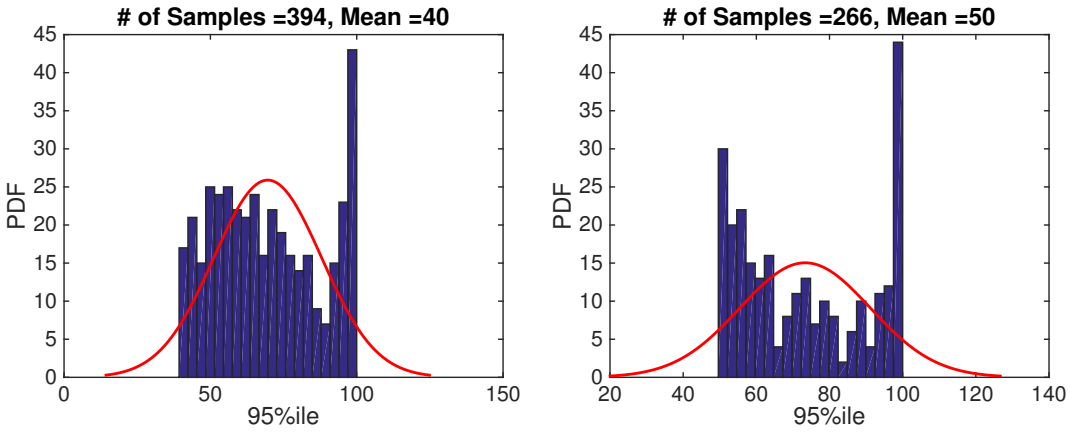
1. We will model the trade-off between prediction accuracy (expensive temporal models) and computational cost (cheap spatial models). More specifically, we will quantify how more reduction in series may affect the accuracy of prediction.

2. We will propose different methods of signature selection, which aim to choose the best candidate to represent the other correlated series. Previous work assumes that the signature series has the highest correlation with all other series in the same cluster. However, this assumption ignores the importance of prediction accuracy from temporal models. We will consider a *temporal-driven* signature selection method and compare with the original method, to find the best signature selection method.

8.2 VM Usage Tail Prediction

Inspired by the box usage tail prediction method proposed in Chapter 7, where we establish that the usage tail follows the normal distribution given the same usage mean, we first directly apply this method to VM usage tails. Surprisingly, this prediction accuracy is

below expectations. We select two different representative distributions of VM usage tails in Figure 8.1. There we observe that VM usage tails do not really follow a normal distribution.



(a) Usage Mean = 40% (b) Usage Mean = 50%

Figure 8.1: The histogram of 95^{th} percentile of VM CPU usage tails with two different usage mean. The bars represent the real distribution of usage tails, and the red line is the fitted normal distribution.

We intend to take the following steps:

1. We will correct the truncated data. From Figure 8.1, we clearly see that the usage tail distribution is truncated at 100. This is because the usage recorded in the traces is no more than 100%. But in this case, the demands from VMs are more than the allocated virtual resource to them. To accurately capture the *real* demands of VMs, we plan to leverage kernel estimation to correct these truncated data in a statistical way.

2. We will focus on fitting the distribution of VM usage tails. As shown in Figure 8.1, usage tails appear to separate into two different regions (states). Based on this observation, we plan to propose a *state-aware* tail usage prediction method, that applies bimodal distribution estimation method to fit VM usage tails, and then uses a Markovian model to capture the state changes of VM usage tails.

8.3 Missing Data in the Wild

In this project, we will concentrate on the problem of missing data in the real-world traces. For plenty of reasons, such as system or sensor failures, it is common for the real-world production systems to have missing data in their log files. In the traces of IBM cloud data centers, almost over 30% of physical boxes experience *time holes* in their VM usage series. In Figure 8.2, we show an example with missing data, where the CPU usage series of three co-located VMs are presented. There we observe that during the first 5 days, the usage series of these three VMs are quite correlated. On the 6th day, VM_3 does not have any recorded usage. However, given the strong correlation among these three VMs, we could fill up the missing data in the usage series of VM_3 based on VM_1 and VM_2.

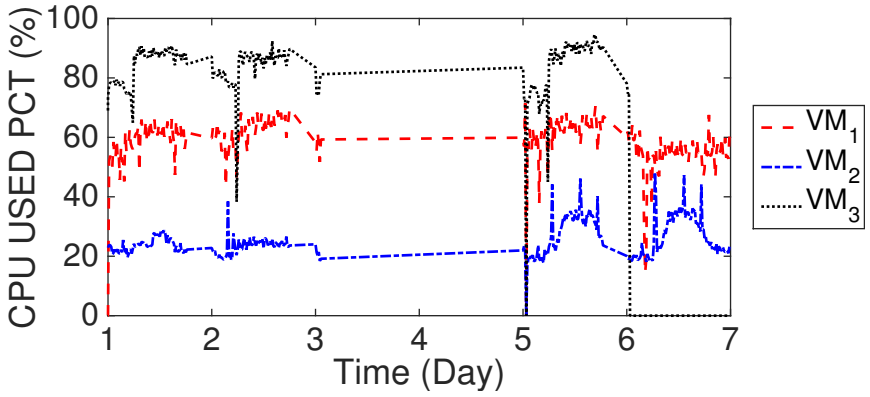


Figure 8.2: CPU usage series of three co-located VMs within the same box.

Motivated by the above example, we propose the following:

1. We first plan to characterize on the usage traces from IBM cloud data centers. More specifically, we will measure the possibility to experience missing data and characterize the dependency within/among the usage series, to explore the potential solution to fill up the missing data, from traces without missing data.

2. We also plan to quantify the dependency among the usage series and to propose a

dependency-driven filling-up method. More specifically, with the spatial-dependency analysis, the series with missing data could be filled-up or predicted using the remaining highly *correlated* and *complete* series. Finally the proposed filling-up method will be evaluated based on the real-world traces.

Bibliography

[1] Amazon ElastiCache. http://aws.amazon.com/elasticache.

[2] CGROUPS. https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.

[3] CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html.

[4] FIO Benchmark. http://www.freecode.com/projects/fio.

[5] Inside the Windows Vista Kernel. http://technet.microsoft.com/.

[6] MediaWiki. https://www.mediawiki.org/wiki/MediaWiki.

[7] HPSS User's Guide, 1995. http://www.hpss-collaboration.org.

[8] Marcos K Aguilera, Kimberly Keeton, Arif Merchant, Kiran-Kumar Muniswamy-Reddy, and Mustafa Uysal. Improving recoverability in multi-tier storage systems. In *Dependable Systems and Networks (DSN), 2007 37th Annual IEEE/IFIP International Conference on*, pages 677–686. IEEE, 2007.

[9] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621, 2010.

[10] Allan T Andersen and Bo Friis Nielsen. A markovian approach for modeling packet traffic with long-range dependence. *IEEE journal on Selected Areas in Communications*, 16(5):719–732, 1998.

[11] Danilo Ansaloni, Lydia Y Chen, Evgenia Smirni, and Walter Binder. Model-driven consolidation of java workloads on multicores. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2012.

[12] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the 1st Annual ACM SIGMM Conference on Multimedia Systems*, pages 35–46. ACM, 2010.

[13] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370, 1994.

[14] Robert Birke, Mathias Björkqvist, Lydia Y Chen, Evgenia Smirni, and Ton Engbersen. (Big)data in a virtualized world: volume, velocity, and variety in cloud datacenters. In *FAST*, pages 177–189, 2014.

[15] Robert Birke, Ioana Giurgiu, Lydia Y Chen, Dorothea Wiesmann, and Ton Engbersen. Failure analysis of virtual and physical machines: patterns, causes and characteristics. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2014.

[16] Robert Birke, Andrej Podzimek, Lydia Y Chen, and Evgenia Smirni. State-of-the-practice in data center virtualization: Toward a better understanding of vm usage. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2013.

[17] Mathias Björkqvist, Lydia Y Chen, and Walter Binder. Opportunistic service provisioning in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 237–244. IEEE, 2012.

[18] Thomas Bonald and James Roberts. Multi-resource fairness: Objectives, algorithms and performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 43, pages 31–42. ACM, 2015.

[19] Mirela Madalina Botezatu, Jasmina Bogojeska, Ioana Giurgiu, Hagen Voelzer, and Dorothea Wiesmann. Multi-view incident ticket clustering for optimal ticket dispatching. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1711–1720. ACM, 2015.

[20] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[21] Giuliano Casale. Building accurate workload models using markovian arrival processes. In *Proceedings of the ACM SIGMETRICS joint International Conference on Measurement and Modeling of Computer Systems*, pages 357–358. ACM, 2011.

[22] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review*, 35(5):103–116, 2001.

[23] Chris Chatfield. *The analysis of time series: an introduction*. CRC press, 2013.

[24] Jiongze Chen, Ronald G Addie, and Moshe Zukerman. Performance evaluation and service rate provisioning for a queue with fractional brownian input. *Performance Evaluation*, 70(11):1028–1045, 2013.

[25] Peter M Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The rio file cache: Surviv-

ing operating system crashes. In *ACM SIGPLAN Notices*, volume 31, pages 74–83. ACM, 1996.

[26] Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *Dependable Systems and Networks (DSN), 2008 38th Annual IEEE/IFIP International Conference on*, pages 452–461. IEEE, 2008.

[27] Hyung Won Choi, Hukeun Kwak, Andrew Sohn, and Kyusik Chung. Autonomous learning for efficient resource utilization of dynamic vm migration. In *Proceedings of the 22nd annual International Conference on Supercomputing*, pages 185–194. ACM, 2008.

[28] Edwin Chuah, Arshad Jhumka, Sai Narasimhamurthy, John Hammond, James C Browne, and Bill Barth. Linking resource usage anomalies with system failures from cluster log data. In *Reliable Distributed Systems (SRDS), 2015 IEEE 34th Symposium on*, pages 111–120. IEEE, 2013.

[29] Javier Alvarez Cid-Fuentes, Claudia Szabo, and Katrina Falkner. Online behavior identification in distributed systems. In *Reliable Distributed Systems (SRDS), 2015 IEEE 34th Symposium on*, pages 202–211. IEEE, 2015.

[30] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI*, pages 273–286, 2005.

[31] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M Hellerstein, and Caleb Welton. Mad skills: new analysis practices for big data. *Proceedings of the 35th International Conference on Very Large Data Bases*, 2(2):1481–1492, 2009.

[32] Paulo Cortez, Miguel Rio, Miguel Rocha, and Pedro Sousa. Multi-scale internet traffic forecasting using neural networks and time series methods. *Expert Systems*, 29(2):143–155, 2012.

[33] Maria Couceiro, Paolo Romano, and Luis Rodrigues. A machine learning approach to performance prediction of total order broadcast protocols. In *Self-Adaptive and Self-Organizing Systems (SASO), 2010 4th IEEE International Conference on*, pages 184–193. IEEE, 2010.

[34] Sven F Crone, Michèle Hibon, and Konstantinos Nikolopoulos. Advances in forecasting with neural networks: Empirical evidence from the nn3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635–660, 2011.

[35] Tommaso Cucinotta, Fabio Checconi, Luca Abeni, and Luigi Palopoli. Self-tuning schedulers for legacy real-time applications. In *Proceedings of the 5th European Conference on Computer Systems*, pages 55–68. ACM, 2010.

[36] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. In *ACM SIGPLAN Notices*, volume 49, pages 127–144. ACM, 2014.

[37] Howard Demuth, Mark Beale, and Martin Hagan. Neural network toolbox^{TM} 6. *User's Guide*, 2008.

[38] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. Enhancing performance prediction robustness by combining analytical modeling and

machine learning. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 145–156. ACM, 2015.

[39] Lars Eggert and Joseph D Touch. Idletime scheduling with preemption intervals. *ACM SIGOPS Operating Systems Review*, 39(5):249–262, 2005.

[40] Ana Juan Ferrer, Francisco HernáNdez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, RaüL Sirvent, Jordi Guitart, Rosa M Badia, Karim Djemame, et al. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.

[41] Ray J Frank, Neil Davey, and Stephen P Hunt. Time series prediction and neural networks. *Journal of Intelligent and Robotic Systems*, 31(1):91–103, 2001.

[42] Rod Fretwell, Demetres Kouvatsos, Engineering Research Group, et al. Lrd and srd traffic: review of results and open issues for the batch renewal process. *Performance Evaluation*, 48(1):267–284, 2002.

[43] Ben D Fulcher and Nick S Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

[44] Paul Massiglia Ganesh Karche, Murthy Mamidi. Using dynamic storage tiering. *Symantec Yellow Books*, 2006.

[45] Box George. *Time Series Analysis: Forecasting & Control, 3rd ed.* Pearson Education India, 1994.

[46] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, volume 11, pages 24–24, 2011.

[47] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 671–678. IEEE, 2013.

[48] Ioana Giurgiu, Adela-Diana Almasi, and Dorothea Wiesmann. Do you know how to configure your enterprise relational database to reduce incidents? In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 339–347. IEEE, 2015.

[49] Ioana Giurgiu, Jasmina Bogojeska, Sergii Nikolaiev, George Stark, and Dorothea Wiesmann. Analysis of labor efforts and their impact factors to solve server incidents in datacenters. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 424–433. IEEE, 2014.

[50] Ioana Giurgiu, Mirela Botezatu, and Dorothea Wiesmann. Comprehensible models for reconfiguring enterprise relational databases to avoid incidents. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1371–1380. ACM, 2015.

[51] Paul Goodwin et al. The holt-winters approach to exponential smoothing: 50 years old and going strong. *Foresight*, 19:30–33, 2010.

[52] Qiang Guan and Song Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *Reliable Distributed Systems (SRDS), 2015 IEEE 34th Symposium on*, pages 205–214. IEEE, 2013.

[53] Jorge Guerra, Himabindu Pucha, Joseph S Glider, Wendy Belluomini, and Raju Rangaswami. Cost effective storage using extent based dynamic tiering. In *FAST*, volume 11, pages 20–20, 2011.

[54] Ajay Gulati, Arif Merchant, and Peter J Varman. pclock: an arrival curve based approach for qos guarantees in shared storage systems. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 13–24. ACM, 2007.

[55] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

[56] Mohamad H Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[57] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, volume 11, pages 261–272, 2011.

[58] Tim Hill, Marcus O'Connor, and William Remus. Neural network models for time series forecasts. *Management Science*, 42(7):1082–1092, 1996.

[59] Kun-Long Ho, Y-Y Hsu, and C-C Yang. Short term load forecasting using a multilayer neural network with an adaptive learning algorithm. *IEEE Transactions on Power Systems*, 7(1):141–149, 1992.

[60] Michael Kutner, Christopher Nachtsheim, and John Neter. *Applied Linear Regression Models*. McGraw-Hill Education, 2004.

[61] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M Rumble, Eyal De Lara, Michael Brudno, and Mahadev Satyanarayanan. Snowflock: rapid virtual machine

cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, pages 1–12. ACM, 2009.

[62] Bob Laliberte. Automate and optimize a tiered storage environment - FAST! *ESG White Paper*, 2009.

[63] Lawrence M Leemis and Stephen Keith Park. *Discrete-event simulation: a first course*. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

[64] Hui Li. Workload dynamics on clusters and grids. *The Journal of Supercomputing*, 47(1):1–20, 2009.

[65] Jia Li and Andrew W Moore. Forecasting web page views: methods and observations. *Journal of Machine Learning Research*, 9(10):2217–2250, 2008.

[66] Ming Li, Deepak Ganesan, and Prashant Shenoy. Presto: Feedback-driven data management in sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1256–1269, 2009.

[67] Yan Li, Nakul Sanjay Dhotre, Yasuhiro Ohara, Thomas M Kroeger, Ethan L Miller, and Darrell DE Long. Horus: fine-grained encryption-based security for large-scale storage. In *FAST*, pages 147–160, 2013.

[68] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Morris Jette, and Ramendra Sahoo. Bluegene/l failure analysis and prediction models. In *Dependable Systems and Networks (DSN) 2006 36th Annual IEEE/IFIP International Conference on*, pages 425–434. IEEE, 2006.

[69] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated control for elastic storage. In *Proceedings of the 7th International Conference on Autonomic Computing*, pages 1–10. ACM, 2010.

[70] Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, and Chen Yu. Live migration of virtual machine based on full system trace and replay. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, pages 101–110. ACM, 2009.

[71] Haikun Liu, Hai Jin, Cheng-Zhong Xu, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. *Cluster Computing*, 16(2):249–264, 2013.

[72] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

[73] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *ACM Sigplan Notices*, volume 43, pages 329–339. ACM, 2008.

[74] Fumio Machida, Dong Seong Kim, and Kishor S Trivedi. Modeling and analysis of software rejuvenation in a server virtualized system with live vm migration. *Performance Evaluation*, 70(3):212–230, 2013.

[75] Tara M. Madhyastha and Daniel A. Reed. Learning to classify parallel input/output access patterns. *IEEE Transactions on Parallel and Distributed Systems*, 13(8):802–813, 2002.

[76] Amiya K Maji, Subrata Mitra, Bowen Zhou, Saurabh Bagchi, and Akshat Verma. Mitigating interference in cloud services by middleware reconfiguration. In *Proceedings of the 15th International Conference on Middleware*, pages 277–288. ACM, 2014.

[77] Morris L Marx and Richard J Larsen. *Introduction to mathematical statistics and its applications*. Pearson/Prentice Hall, 2006.

[78] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[79] Peter Membrey, Eelco Plugge, and David Hows. *Practical Load Balancing*. Apress, 2012.

[80] Madalin Mihailescu, Gokul Soundararajan, and Cristiana Amza. Mixapart: decoupled analytics for shared storage systems. In *FAST*, pages 133–146, 2013.

[81] Luca Muscariello, M Meillia, Michela Meo, Marco Ajmone Marsan, and Renato Lo Cigno. An mmpp-based hierarchical model of internet traffic. In *Communications, 2004 IEEE International Conference on*, volume 4, pages 2143–2147. IEEE, 2004.

[82] Dushyanth Narayanan, Austin Donnelly, Eno Thereska, Sameh Elnikety, and Antony IT Rowstron. Everest: Scaling down peak loads through I/O off-loading. In *OSDI*, volume 8, pages 15–28, 2008.

[83] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast transparent migration for virtual machines. In *USENIX Annual Technical Conference, General Track*, pages 391–394, 2005.

[84] Marcel F Neuts. *Structured stochastic matrices of MG-1 type and their applications*. Dekker, 1989.

[85] Adrian Nistor, Po-Chun Chang, Cosmin Radoi, and Shan Lu. Caramel: detecting and fixing performance problems that have non-intrusive fixes. In *Proceed-*

ings of the 37th International Conference on Software Engineering-Volume 1, pages 902–912. IEEE Press, 2015.

[86] Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H Noh. Caching less for better performance: balancing cache size and update cost of flash memory cache in hybrid storage systems. In *FAST*, volume 12, 2012.

[87] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, et al. The case for ramclouds: scalable high-performance storage entirely in dram. *ACM SIGOPS Operating Systems Review*, 43(4):92–105, 2010.

[88] Roxy Peck, Chris Olsen, and Jay Devore. *Introduction to statistics and data analysis*. Cengage Learning, 2015.

[89] Mark Peters. 3PAR: Optimizing I/O service levels. *ESG White Paper*, 2010.

[90] David Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83(2):394–410, 1995.

[91] Lior Rokach and Oded Maimon. Clustering methods. *Data mining and knowledge discovery handbook*, pages 321–352, 2005.

[92] Andrea Rosa, Lydia Y Chen, and Walter Binder. Understanding the dark side of big data clusters: an analysis beyond failures. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 207–218. IEEE, 2015.

[93] Sheldon M Ross. *Introduction to probability and statistics for engineers and scientists*. Academic Press, 2009.

[94] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[95] Lalit Mohan Saini and Mahender Kumar Soni. Artificial neural network-based peak load forecasting using conjugate gradient methods. *IEEE Transactions on Power Systems*, 17(3):907–912, 2002.

[96] Mohit Saxena, Michael M Swift, and Yiying Zhang. Flashtier: a lightweight, consistent and durable storage cache. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 267–280. ACM, 2012.

[97] Bianca Schroeder, Raghav Lagisetty, and Arif Merchant. Flash reliability in production: The expected and the unexpected. In *FAST*, pages 67–80, 2016.

[98] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. Easyticket: a ticket routing recommendation engine for enterprise problem resolution. *Proceedings of the 34th International Conference on Very Large Data Bases*, 1(2):1436–1439, 2008.

[99] Phlip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. Wan-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage (TOS)*, 8(4):13, 2012.

[100] Rahul Singh, Prashant Shenoy, Maitreya Natu, Vaishali Sadaphal, and Harrick Vin. Analytical modeling for what-if analysis in complex cloud computing applications. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):53–62, 2013.

[101] Simon Spinner, Nikolas Herbst, Samuel Kounev, Xiaoyun Zhu, Lei Lu, Mustafa Uysal, and Rean Griffith. Proactive memory scaling of virtualized applications. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 277–284. IEEE, 2015.

[102] Avinash Sridharan and Bhaskar Krishnamachari. Maximizing network utilization with max–min fairness in wireless sensor networks. *Wireless Networks*, 15(5):585–600, 2009.

[103] Murray Stokely, Amaan Mehrabian, Christoph Albrecht, Francois Labelle, and Arif Merchant. Projecting disk usage based on historical trends in a cloud environment. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pages 63–70. ACM, 2012.

[104] Ajay Surie, H Andrés Lagar-Cavilla, Eyal de Lara, and Mahadev Satyanarayanan. Low-bandwidth vm migration via opportunistic replay. In *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications*, pages 74–79. ACM, 2008.

[105] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, and Deepak Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 285–294. IEEE, 2012.

[106] Leandros Tassiulas and Saswati Sarkar. Maxmin fair scheduling in wireless networks. In *INFOCOM*, volume 2, pages 763–772. IEEE, 2002.

[107] Henk C Tijms. *A first course in stochastic models*. John Wiley and Sons, 2003.

[108] Devesh Tiwari, Simona Boboila, Sudharshan S Vazhkudai, Youngjae Kim, Xiaosong Ma, Peter Desnoyers, and Yan Solihin. Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In *FAST*, pages 119–132, 2013.

[109] Nancy Tran and Daniel A Reed. Automatic arima time series modeling for adaptive i/o prefetching. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):362–377, 2004.

[110] Graham Upton and Ian Cook. *A Dictionary of Statistics 3e*. Oxford university press, 2014.

[111] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830–1845, 2009.

[112] I Var. Multivariate data analysis. *vectors*, 8(2):125–136, 1998.

[113] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *FAST*, volume 12, 2012.

[114] Hui Wang and Peter J Varman. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *FAST*, pages 229–242, 2014.

[115] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM*, pages 71–75. IEEE, 2011.

[116] Timothy Wood, Prashant J Shenoy, Arun Venkataramani, and Mazin S Yousif. Black-box and gray-box strategies for virtual machine migration. In *NSDI*, volume 7, pages 17–17, 2007.

[117] Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. vperfguard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 271–282. ACM, 2013.

[118] Ji Xue, Robert Birke, Lydia Y Chen, and Evgenia Smirni. Managing data center tickets: Prediction and active sizing. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 335–346. IEEE, 2016.

[119] Ji Xue, Robert Birke, Lydia Y Chen, and Evgenia Smirni. Tale of tails: Anomaly avoidance in data centers. In *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*, pages 91–100. IEEE, 2016.

[120] Ji Xue, Feng Yan, Robert Birke, Lydia Y Chen, Thomas Scherer, and Evgenia Smirni. PRACTISE: robust prediction of data center time series. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 126–134. IEEE, 2015.

[121] Ji Xue, Feng Yan, Alma Riska, and Evgenia Smirni. Storage workload isolation via tier warming: How models can help. In *Proceedings of the 11th International Conference on Autonomic Computing*, pages 1–11, 2014.

[122] Ji Xue, Feng Yan, Alma Riska, and Evgenia Smirni. Proactive management of systems via hybrid analytic techniques. In *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, pages 137–148. IEEE, 2015.

[123] Ji Xue, Feng Yan, Alma Riska, and Evgenia Smirni. Scheduling data analytics work with performance guarantees: queuing and machine learning models in synergy. *Cluster Computing*, 19(2):849–864, 2016.

[124] Feng Yan, Alma Riska, and Evgenia Smirni. Busy bee: how to use traffic information for better scheduling of background tasks. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 145–156. ACM, 2012.

[125] Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay Jain, and Michael Stumm. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *OSDI*, pages 249–265, 2014.

[126] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. Encore: Exploiting system environment and correlation information for misconfiguration detection. In *ACM SIGPLAN Notices*, volume 49, pages 687–700. ACM, 2014.

[127] Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of the 4th International Conference on Autonomic Computing*, pages 27–27. IEEE, 2007.

[128] Yiying Zhang, Gokul Soundararajan, Mark W Storer, Lakshmi N Bairavasundaram, Sethuraman Subbiah, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Warming up storage-level caches with bonfire. In *FAST*, pages 59–72, 2013.

[129] Wubai Zhou, Liang Tang, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Resolution recommendation for event tickets in service management. *IEEE Transactions on Network and Service Management*, 13(4):954–967, 2016.

[130] Bonnie Zhu and Shankar Sastry. Revisit dynamic arima based anomaly detection. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE 3rd Inernational Conference on Social Computing (SocialCom), 2011 IEEE 3rd International Conference on*, pages 1263–1268. IEEE, 2011.

[131] Xiaoyun Zhu, Don Young, Brian J Watson, Zhikui Wang, Jerry Rolia, Sharad Singhal, Bret McKee, Chris Hyser, Daniel Gmach, Rob Gardner, et al. 1000 islands: Integrated capacity and workload management for the next generation data center. In *Proceedings of the 8th International Conference on Autonomic Computing*, pages 172–181. IEEE, 2008.

[132] Zhenyun Zhuang, Haricharan Ramachandra, Cuong Tran, Subbu Subramaniam, Chavdar Botev, Chaoyue Xiong, and Badri Sridharan. Capacity planning and headroom analysis for taming database replication latency: experiences with linkedin internet traffic. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 39–50. ACM, 2015.